

Agile Software Development as Managed Sensemaking

by

KOBUS EHLERS

*Thesis presented in fulfilment of the requirements for the degree of Master of
Philosophy at Stellenbosch University*



SUPERVISOR: Dr H.P. Müller
Department of Information Science

March 2011

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

March 2011

Abstract

The environment in which all organisations currently operate is undoubtedly dynamic. Regardless of the nature, size or geographical location of business, companies are being forced to cope with a rapidly changing world and increasing levels of unpredictability.

This thesis tracks the history of software development methodologies leading up to agile development (chapter 2). Agile development has appeared in response to the limitations of traditional development approaches and evolved to address the particular demands of a changing world (chapter 3).

The theory of sensemaking is used to gain insight into the functioning of agile development. Sensemaking is introduced and a working definition of this concept is formulated (chapter 4).

This research does not argue that agile development is the same as sensemaking, but rather that it can be better understood through sensemaking. Agile development can be seen as a type of sensemaking, but sensemaking is also a generic, universal cognitive ability. The structure and design of agile development is well aligned with sensemaking, and one can understand its nature and the type of management needed to support agile development better from this perspective. In fact, agile development directly supports and facilitates several important elements of the sensemaking process.

For successful sensemaking to occur, certain organisational conditions need to be present. The term “managed sensemaking” is introduced to expand this notion.

After performing an analysis of agile development (chapter 5), certain pertinent implications and challenges facing organisations are considered (chapter 6). By framing these implications in terms of sensemaking, practical management suggestions can be provided based on a good fit between the problem that agile development is meant to solve and the cognitive requirements of the process leading to a solution.

The research conducted in this process opens the door to further research opportuni-

ties (chapter 7) and allows for the application of sensemaking in the context of software development methodologies.

This study provides insight into the prevalence and functioning of agile methodologies, in software engineering contexts, by leveraging the theory of sensemaking to provide an explanation for the underlying worldview and processes constituting this approach.

Opsomming

Die omgewing waarin alle organisasies tans funksioneer in ongetwyfeld dinamies. Maatskappye word genoep om die uitdagings van 'n vinnig-veranderende wêreld die hoof te bied, ongeag die aard, grootte of geografiese ligging van die besigheid.

Hierdie tesis volg die geskiedenis van sagteware-ontwikkelingsmetodologië tot by *agile development* (hoofstuk 2). *Agile development* het verskyn as 'n reaksie op die beperkings van tradisionele ontwikkelingsbenaderings en evolueer om aan te pas by huidige uitdagings (hoofstuk 3).

Die teorie van *sensemaking* word gebruik om insig te verkry in die funksionering van *agile development*. *Sensemaking* word ingelei en 'n werksdefinisie word geformuleer (hoofstuk 4).

Hierdie navorsing argumenteer nie dat *agile development* dieselfde is as *sensemaking* nie, maar eerder dat dit beter verstaan kan word deur *sensemaking*. *Agile development* kan wél gesien word as 'n tipe *sensemaking*, maar *sensemaking* is ook 'n generiese, universieële kognitiewe vermoë. Die struktuur en ontwerp van *agile development* is goed belyn met *sensemaking*, en 'n mens kan die aard daarvan en tipe bestuur benodig om *agile development* te ondersteun beter verstaan vanuit hierdie perspektief. Tewens, *agile development* ondersteun en fasiliteer verskeie belangrike elemente van die *sensemaking* prosesse direk.

Vir suksesvolle *sensemaking* om plaas te vind, word sekere organisatoriese toestande benodig. Die term “*managed sensemaking*” word ingelei om hierdie idee uit te brei.

Ná 'n analise van *agile development* (hoofstuk 5) word sekere dwingende implikasies en uitdagings, wat organisasies in die gesig staar, oorweeg (hoofstuk 6). Deur hierdie implikasies te plaas in *sensemaking*-terme kan praktiese bestuursvoorstelle aangebied word, gegrond op 'n goeie passing tussen die probleem wat *agile development* probeer aanspreek en die kognitiewe vereistes van die prosesse wat lei na 'n oplossing.

Die navorsing wat onderneem is in hierdie prosesse ontsluit moontlikhede vir verdere studies (hoofstuk 7) en skep die moontlikheid vir die toepassing van *sensemaking* in die

konteks van sagtewareontwikkelingsmetodologieë.

Hierdie studie bied insig in die voorkoms en funksionering van *agile methodologies* in sagteware-ingenieurwese omgewings deur die teorie van *sensemaking* te hefboom om 'n verduideliking vir die onderliggende wêreldbeeld en prosesse aan te bied.

Acknowledgements

Completing this thesis would not have been possible without the help and support of many wonderful people. Although it is impossible to list everyone, I would like to thank the following people in particular.

Firstly, I owe a great debt of gratitude to my supervisor, Dr Hans Müller, for his help and guidance. His commitment and attention to detail (regardless of the time pressure) has certainly greatly improved the quality of this thesis. Interacting with him in this context has been nothing but a pleasure.

Secondly, I would like to thank my family for their continued support. Especially my parents deserve enormous thanks for offering me the opportunity of higher education.

Next I would like to acknowledge my colleagues at the Department of Information Science. I really enjoy working at this department and they have definitely contributed to enhancing this experience. A special word of thanks goes to the departmental chair, Prof Johann Kinghorn for his encouragement and insight. I also want to acknowledge all the great lecturers and professors who taught at Stellenbosch University during my studies and sent me down this path.

Then I would be remiss if I did not mention the wonderful support from all my friends during the completion of this degree. Without them this endeavour would have been dull and boring. You are too many to mention, but know that I appreciated all your support. Special mention goes to my housemates (André and Linsen) as well as #VONLAAF. Linsen Loots also deserves enormous thanks for his assistance in the proof reading of this thesis and beautiful redrawing of the figures in this document.

Lastly, I wish to thank the Harry Crossley Foundation and University of Stellenbosch who made this research possible through scholarships and bursaries.

Contents

Nomenclature	xi
1 Introduction	1
1.1 Background and context	1
1.1.1 Information Systems Development Methodologies	4
1.2 Agile Development as Managed Sensemaking	5
1.2.1 Why Sensemaking?	5
1.2.2 Managed Sensemaking	6
1.3 Organisation of this Thesis	7
2 History of Software Methodologies	9
2.1 Terminology & Definitions	9
2.1.1 Software Engineering	10
2.1.2 Methodology	10
2.2 Meta-methodological nature	11
2.3 Historical evolution	12
2.3.1 Pre-methodology era	13
2.3.2 Early-methodology era	15
2.3.3 Methodology era	18
2.3.4 Era of methodology reassessment	21
2.3.5 Age of agility	27
3 Agile Development	28
3.1 Establishment of the ‘Agile Development Movement’	28
3.2 Background	30
3.2.1 Central themes	31

3.3	Summative Description	35
3.4	Prevalence of Agile Development	35
3.5	Scrum	36
3.5.1	Process	38
3.5.2	Roles	40
3.5.3	Artefacts	41
3.5.4	Ceremonies	44
3.6	eXtreme Programming	46
3.6.1	Process	47
3.6.2	Values, Principles and Practices	49
4	Sensemaking	53
4.1	Conceptualising Sensemaking	53
4.1.1	Historical Roots	55
4.2	The Nature of Sensemaking	56
4.2.1	Interpretation vs. Authoring	56
4.2.2	The Sensemaking Activity	58
4.2.3	Seven properties of Sensemaking	59
4.2.4	Characteristics of Sensemaking	71
4.3	Sensemaking as Enactment	76
4.4	Occasions for Sensemaking	77
4.4.1	Ambiguity	78
4.4.2	Uncertainty	79
4.5	Level of Analysis (Institutional Theory)	80
4.5.1	Levels of Sensemaking	80
5	Agile Development as Managed Sensemaking	83
5.1	Meta-sensemaking & Recursion	84
5.1.1	Level of Analysis	84
5.1.2	Distinctions	85
5.1.3	Occasions for Agile Development	86
5.2	Worldview (<i>Weltanschauung</i>)	87
5.2.1	De-emphasising the plan	87

5.2.2	Functional pragmatism	88
5.2.3	Chaos	89
5.3	Creating Order	92
5.4	Agile Development as Process	93
5.4.1	Active Construction Process	93
5.4.2	Ongoing Enactment	94
5.4.3	Retrospection	95
5.4.4	Adaptive Presumptions	96
5.5	Agile Development as Social Activity	97
5.5.1	Relation to Individual Sensemaking	97
5.5.2	Agile Development as Inherently Social	97
5.5.3	Identity Construction	99
5.5.4	Organisation through Communication	100
6	Implications for the Organisation	105
6.1	Management Challenges	106
6.2	Power and Hierarchy	108
6.3	People Issues	109
6.4	Knowledge Management Dimensions	111
6.5	Compliance and Regulation	113
6.6	Coping with these Challenges	114
7	Conclusion	115
7.1	Course of the Research	116
7.2	Conclusions	118
7.3	Further Research	119
7.4	Final Remarks	120
	Bibliography	121

List of Figures

2.1	The Classic Representation of the Waterfall Model	17
2.2	The Spiral Model	20
3.1	The Planning Spectrum	31
3.2	The Scrum Development Process	38
3.3	A Burndown Chart	43
3.4	XP Life-Cycle (Traditional View)	46
3.5	XP Life-Cycle (Expanded View)	47
4.1	The Relationship Among Enactment, Organising, and Sensemaking	76

List of Tables

2.1	David Parnas's list of current software problems	14
2.2	Description of main agile development methods, with key references	26
3.1	Twelve principles of agile development	29
3.2	Principles of Extreme Programming	50
3.3	Practices of Extreme Programming	52
4.1	Characteristics of Ambiguous, Changing Situations	79
5.1	General characteristics of complex systems	91

Nomenclature

Acronyms & Abbreviations

ASD	Adaptive Software Development
BCS	Battered-Child Syndrome
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
CRM	Customer Relationship Management system
DSDM	Dynamic Software Development Method
ERP (1)	Enterprise Resource Planning system
ERP (2)	Enterprise Resource Packages
FDD	Feature Driven Development
IEEE	Institute of Electrical and Electronics Engineers
IID	Iterative Incremental Development
IS	Information Systems
LSD	Lean Software Development
NATO	North Atlantic Treaty Organization
OOP	Object-oriented Programming
OSSD	Open Source Software Development
RAD	Rapid Application Development
RUP	Rational Unified Process
SDLC	Systems Development Life Cycle
SE	Software Engineering
SSM	Soft Systems Methodology
SWEBOK	Software Engineering Body of Knowledge
XP	Extreme Programming

Chapter 1

Introduction

1.1 Background and context

The environment in which all organisations currently operate is undoubtedly dynamic. Regardless of the nature, size or geographical location of business, companies are being forced to cope with a rapidly changing world and increasing levels of unpredictability. This reality impacts all levels of the organisation; from the coal face right up to strategic decision making and planning.

Organisations have developed a myriad of methodologies to plan and execute projects in this new, dynamic environment. Most organisations will have set procedures to deal with almost any eventuality encountered during their normal operations. Although these procedures differ depending on the goal, history, culture and type of organisation, these *traditional* approaches all assume some level of predictability and therefore regularity.

In the recent past we have seen that these *traditional* approaches cannot sufficiently handle an environment that is unpredictable and constantly changing. Most of these approaches are not well suited to deal with complexity and rely on linear causality to explain cause-effect relations. This complexity, compounded by the effects of the knowledge economy and globalisation, have resulted in many organisations redesigning their core business processes or even their entire business offering.

In response to these failings, companies have adjusted their way of looking at the world and organising themselves. A very interesting development is the establishment of more flexible approaches to the planning and execution of projects. This newfound flexibility can

be seen in approaches such as *lean engineering* or *just-in-time* manufacturing¹.

The specific phenomenon investigated in this thesis is Agile Development. This development methodology deviates from traditional software development methodologies such as the *Waterfall Model*. The focus shifts from pre-planning to a much more responsive and dynamic iterative development process employing self-organising and cross-functional teams.

This makes agile development a current and relevant issue for engineering and technology firms. The application of this approach is, however, not limited only to these high-tech companies.

One of the major changes in the modern knowledge economy is the undisputed importance of information systems. Successful organisations spend enormous amounts of time and energy implementing these systems². A variety of electronic tools allows companies to keep track of inventory, monitor processes and support management and decision making throughout the organisation.

Information systems are used for a wide variety of purposes and applications (Laudon & Laudon, 2010) including, but not limited to, Transaction processing systems, Management information systems, Decision support systems and Executive information systems. Their applications have been expanded to (amongst others)³:

- Data warehouses
- Enterprise resource planning
- Enterprise systems
- Expert systems
- Geographic information systems
- Global information systems
- Office automation

¹These theories were pioneered by Taiichi Ohno (1982; 1986; 1988) and others with the classic example being the redesign of production systems at Toyota.

²*Information systems* in our context refer to formalised computer-based systems used to provide processes and information useful to their members and clients.(Avison & Fitzgerald, 2003)

³Please see Laudon & Laudon (2004) for a detailed discussion.

Organisational information systems come in a number of shapes and sizes. Many of these systems are custom built for or by the organisations requiring them, but quite often companies will also buy existing software products from external vendors⁴ or deploying packaged open-source products. Purchasing vendor products is especially popular with large and complicated systems such as ERP's (Enterprise Resource Planning systems) or CRM's (Customer Relationship Management systems), which would be too expensive or time consuming to develop internally. Even with purchased solutions, substantial time and money is allocated to allow for the customisation of the purchased products and their integration with existing business and legacy systems.

This great demand for customisation or novel system development means that most organisations, even those not in a high-tech field, do substantial amounts of software development. In effect most large organisations are being confronted with the failings of traditional project management and development methodologies and many are considering the application of Agile Development practices (or have already implemented them).

The appeal for more flexible development practices was also precipitated by the *software crisis*. This “crisis” refers to the fact that software programming became increasingly difficult as hardware evolved to allow for more complicated software programs. Writing these large, complicated software programs brought about a variety of problems including quality issues, budget over-runs, missed deadlines and complete failures.⁵

The natural organisational response to these problems has been to increase the focus on planning and preparation and prepare for contingencies before they occur. Although a more structured approach certainly had its benefits, pre-planning longer term projects and fixing their design is very problematic when the requirements and environment of the system is constantly changing.

The limitations and frustrations of formal planning approaches lead to the development of more flexible, or *agile*, methodologies for software development. These methodologies place the focus on the product, rather than on extensive planning. The idea is that the product development process becomes more flexible and adaptive and therefore becomes directly sensitive to customer needs, rather than some formal specification (Beck et al.,

⁴These vendors are most often well-known companies such as SAP, Oracle, Sage and Microsoft.

⁵The term “software crisis” was coined by F.L. Bauer during the first NATO Conference on Software Engineering in 1968 (Randell & Naur, 1968).

2001).

The starting point for this study is the prevalence of these new-style methodologies in practice, not only in small start-up software firms (although it is particularly popular there), but also in larger organisations facing the challenges of developing enterprise software in the current market environment.

1.1.1 Information Systems Development Methodologies

Information systems development methodologies are normally seen as the tools, techniques and procedures required by developers to complete their assigned tasks (Avison & Fitzgerald, 2003). The focus is a pragmatic one - a selection of items that aid the developer in reaching his/her goal. Many IS researchers will, however, concede that there is a specific philosophy attached to a methodology. This distinguishes a *methodology* from mere *method*.

The British Computer Society (BCS) Information Systems Analysis and Design Working Group in 1983 released a definition for *information systems methodology* today still considered a seminal definition (Maddison, 1983):

a recommended collection of philosophies, phases, procedures, rules, techniques, tools, documentation, management and training for developers of information systems.

Traditional definitions make it clear that methodologies are regarded as a technical concern that influences the way in which programmers and developers do their jobs. While some methodologies may cater for issues such as power and politics or human factors, the rationale behind using a specific set of methodologies remains to improve the development process of the software product.

Jayaratna (2004, p. 37) points out that methodologies provide an “explicit way of structuring” development:

Methodologies contain models and reflect particular perspectives of ‘reality’ based on a set of philosophical paradigms. A methodology should tell you ‘what’ steps to take and ‘how’ to perform those steps but most importantly the reasons ‘why’ those steps should be taken, in that particular order.

Software development methodologies play an important role in modern organisations

reliant on a variety of information systems. This study focuses on better understanding agile development as a markedly new approach to software development.

1.2 Agile Development as Managed Sensemaking

Many organisations of varying sizes are all faced with the reality of coping with agile development as a new and distinctly different approach to the old problem of software development⁶. A large portion of these organisations are not accustomed to practices proposed by this new approach and operate with a vastly different worldview to that assumed by agile development. To provide a better understanding of this novel approach, this thesis will leverage sensemaking theory to give insight into and enable an organisational analysis of agile development.

In this thesis it will be argued that the selection and use of agile development methodologies impact far more than merely the programming style and development techniques of the organisation concerned. The use of agile development methodologies will be investigated as an activity of organisational sensemaking and identity construction.

1.2.1 Why Sensemaking?

Sensemaking is an attempt to understand how “active agents construct sensible, sensible events” (Huber & Daft, 1987). Sensemaking is especially useful in trying to understand how organisations deal with ambiguous or uncertain situations. These are the type of situations where predictions break down and stimuli are put into frameworks (Louis, 1980).

Since agile development is deployed in exactly these situations of uncertainty, the methodology (or rather *methodologies*) itself is also suited for analysis as an instance of sensemaking. The predominant author on the topic of organisational sensemaking is Karl Weick. This text will try to analyse the phenomenon of agile development as an information systems development methodology using his model and frameworks for sensemaking.

Weick’s (1995) formulation of sensemaking will be expanded by the work done by Dervin (1983; 1992; 1999) on sensemaking in the information systems context.

⁶It should however be noted that although Agile Development is definitely not the same as traditional software development methodologies, it progressed from several iterative developments in software engineering that will be considered later.

Traditionally, software engineering has been viewed by outsiders in the same terms as classical engineering or computer science. The move towards agile approaches or more “lightweight” methodologies has, however, resulted in a new focus on the human engineers and their beliefs and values. Sensemaking also has the necessary sensitivity to and awareness of these issues to allow for a sensible analysis.

This study aims to investigate agile development as a sensemaking activity and to gain further insight by viewing these methodologies from a sensemaking perspective. Additionally the impact of agile methodologies on organisations are considered.

This research does not argue that agile development is the same as sensemaking, but rather that agile development can be better understood through sensemaking. Agile development can be seen as a type of sensemaking, but sensemaking is also a generic, universal cognitive ability. The structure and design of agile development is well aligned with sensemaking, making it more suitable for analysis from this perspective than more traditional software development methodologies such as the Waterfall Model. Additionally, agile development supports and facilitates several important elements of the sensemaking process.

Traditional methodologies could of course also be understood from a sensemaking perspective. If one were to complete this analysis one would be able to see how the constraints of traditional methodologies limit and restrict sensemaking practice. This is, however, outside the scope of this study. The focus here is rather to use sensemaking to *explain* agile development, its adoption and the reports of benefit to organisations deploying this methodology.

1.2.2 Managed Sensemaking

This thesis refers to sensemaking in a very specific way, here called “managed sensemaking”. Chapter 4 deals with sensemaking itself in some detail, but in the context of this analysis of agile development we are focusing on more than the generic, universal, cognitive process called sensemaking.

The term “managed sensemaking” is used because the conditions and structure required to enable and encourage sensemaking need to be managed. Sensemaking (as present in agile development methodologies) can only flourish if the organisation lets it. Organisations need to “let go” and avoid resisting change and uncertainty. By trusting the sensemaking process and not restricting learning, organisations can create the conditions (or “manage the

interfaces”) required for effective sensemaking. This is not the default behaviour found in organisations and therefore requires active management and also a change in management style and the conceptualisation of the purpose or role of management.

At the same time this term should not be understood to mean that the sensemaking process *itself* is being managed. From later discussions it will be clear that it is not only an unwanted interference to try to manage the sensemaking process, but also a futile attempt.

1.3 Organisation of this Thesis

This thesis investigates agile development as a process of sensemaking that is managed or conscious in a sense that will be explained later. Following this very broad introduction, it will be structured as follows:

Chapter 2 starts by looking at software engineering and the history and evolution of software development methodologies. This chapter gives a chronological and thematic review of the organisation of software development up to the introduction of “agile development”.

Chapter 3 discusses the concept “agile development”. One of the first challenges in writing about this methodology is the lack of good quality academic literature. This shortcoming is explained by the relative youth of this approach. It is also compounded by the pragmatic (documentation averse) nature of the methodology. Although there is general agreement that there is a class of methodologies that can be described as “agile”, an authoritative definition of this class is lacking. The salient characteristics of this approach are distilled and a working definition constructed for use in the rest of this thesis.

Chapter 4 is a brief summary of Weick’s description of sensemaking expanded by influences from several leading authors in this field. Since Weick does not provide a definitive model (or set of criteria or checklists) with which to evaluate phenomena for their sense-making properties, the description of sensemaking is adapted to allow for its use in the analysis of agile development. It is important to note that *sensemaking* here refers to both the generic, universal, cognitive act performed by individuals and also the higher-level processes occurring during organisational sensemaking.

Chapter 5 then proceeds to integrate (or “synthesise”) the notions of agile development and sensemaking. By viewing agile development through the lens of sensemaking theory, additional insight can be gained into the this phenomenon. Additionally, sensemaking can

be used to understand the world view giving rise to the development of agile methodologies. This chapter considers practical elements of agile methodologies and analyses them from a sensemaking perspective.

Chapter 6 considers several implications organisations are faced with when deploying agile development processes. Properly understanding these challenges is paramount to the successful deployment or integration of these novel development approaches. The conceptualisation of agile development as a sensemaking activity unlocks certain response options in companies dealing with new and emergent organisational structures.

Chapter 7 concludes the research and delivers some general commentary. Consideration is given to the practical use of the findings and some areas for further research are identified.

Chapter 2

History of Software Methodologies

This thesis deals with agile development as a specific software engineering approach. Simply investigating the phenomenon as a contingent fact will, however, miss some of the important events that led to the development of this approach. Agile development is situated in history as a reaction to previous software engineering methodologies - especially as a reaction to “plan-based or traditional methods” (Larman & Basili, 2003) with their focus on “a rationalized, engineering-based approach” (Nerur et al., 2005; Dyba, 2000).

A basic knowledge of this antithetical nature and historical context is essential not only for a proper understanding of agile development, but also important for the sensemaking analysis that will follow later in this study.

The evolution of software engineering from so-called *traditional* approaches to what we now call *agile* approaches should not be seen as a simple linear development and may have followed several paths. To this end, the phenomenon is investigated chronologically *and* thematically. This introduction will provide the necessary context for understanding the detail of agile development discussed in the next chapter.

Although agile development is a newer approach, this does not mean that all organisations have moved (or will move) to employing these methodologies. It is simply one type of approach that can be applied towards organising software development.

2.1 Terminology & Definitions

The issues dealt with in this thesis are decidedly practical. As is indicated throughout this text, agile development was developed by practitioners of software engineering for

use in industry products, rather than as a theoretical or academic construct (Ågerfalk & Fitzgerald, 2006). As a result, the terminology used when describing agile development is not as exact as would be expected in an academic environment.

To try and alleviate the confusion, some of these variable uses are indicated in the sections that follow.

2.1.1 Software Engineering

Software Engineering as a concept was defined at the NATO Science Conference of 1968. This conference was called to address the so-called *software crisis* (Randell & Naur, 1968) discussed later in this chapter.

Subsequently many definitions have followed and the original concept has also changed to represent the contemporary use of this term. The definition used in this thesis is provided by the *IEEE Computer Society’s Software Engineering Body of Knowledge (SWEBOK)*¹:

the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software.

As can be gleaned from this definition, software engineering refers not simply to the act of programming, but to the entire development process. In this thesis the terms *software engineering* and *software development* are used interchangeably.

2.1.2 Methodology

We have already referred to the BCS definition for *information system methodology* in the first chapter². This definition is expanded by Avison and Fitzgerald³ to include the notion of “philosophy” and “beliefs” to read

A system development methodology is a recommended means to achieve the development, or part of the development, of information systems based on a set of rationales and an underlying philosophy that supports, justifies and makes

¹SWEBOK in turn takes its definition from the IEEE Standard Glossary of Software Engineering Terminology (IEEE, 1990)

²p. 4

³Avison & Fitzgerald (2003, p. 24)

coherent such a recommendation for a particular context. The recommended means usually includes the identification of phases, procedures, tasks, rules, techniques, guidelines, documentation and tools. They might also include recommendations concerning the management and organisation of the approach and the identification and training of the participants.

We should distinguish between *methodology* and *method*, with the former incorporating the entire set of beliefs and philosophical assumptions that accompany the “means of development” (as noted by the definition). In contrast, method refers simply to the “way of doing”. Avison & Fitzgerald (2003) also describe this a *technique*.

Although these distinctions are important in certain contexts, one should bear in mind that the cited literature was most often written for an industry audience, leading to a relaxed use of these terms. Where the distinction is important it will be indicated.

There is also a certain amount of overlap between the different uses of these terms. For example, the term “waterfall model” could refer both to a *software development* methodology and a *project management* methodology.

The undefined nature of agile development as a methodology (or, as is argued here, something more) means that the terms methodology, method, approach, style and others may all be used to refer to *agile development*. Care will be taken to avoid confusion in these cases.

Agile development can also not be described as a single, formal methodology and therefore both the terms *methodology* and *methodologies* are used when referring to the phenomenon.

2.2 Meta-methodological nature

Not surprisingly, some of the newer style (emergent) methodologies display some self-similar characteristics in their ‘design’ and development. Many of the reflexive development principles promoted by light-weight approaches are also applied to the conceptualisation of these approaches themselves.

In the case of agile development one of the central tenets (discussed in the next chapter) is that documentation should be kept to a minimum. This mantra has also been applied in the meta-methodological dealings of the Agile Alliance and other proponents. Since

agile development was not formally planned and designed, but evolved organically, the documentation relating to this practice is also limited.

The prevalence of these methodologies has, however, meant that quite a few studies have been conducted, in addition to the guides and tools developed by the proponents of these approaches.

Agile development methodologies are applied both in a normative and a descriptive mode and the distinction between these functions is as important in this study as it is in practice.

This thesis evaluates both the defined normative prescriptions offered by several authors of agile methodologies, and the descriptive academic and industry studies into the use and effects of these methodologies. No claim is made that this study will provide any final definition for agile development⁴, but through the historical and functional analyses in this chapter and the next, enough synthesis will be done to provide a workable understanding of what we refer to as “agile development”.

2.3 Historical evolution

In order to understand the process and events leading up to the establishment of agile development methodologies, this section will outline some of the historical developments in software engineering⁵. Although this historical account is presented in five reasonably chronological sections, it is important to remember that (like most developments) methodological changes were not a simple linear process.

There are also interesting questions to be asked about the nature of these changes and whether they were *revolutionary* or *evolutionary*⁶. For the purposes of this thesis we simply focus on the fact that methodologies *changed* over time and that agile development methodologies have a certain heritage.

To discuss the historical evolution of software methodologies, we will build on the three-stage approach developed by Avison & Fitzgerald (2003) and expand it by adding two new

⁴This is also not an attempt to try and provide some final definition applicable to all contexts.

⁵Several accounts of the evolution of software methodologies have already been published. This chapter aims to summarise the points relevant to this study.

⁶For an excellent discussion of this topic, see Northover (2008).

stages.

2.3.1 Pre-methodology era

The first stage we will look at was characterised by a very *ad hoc* and informal way of developing information systems. No formal methodologies or explicit procedures were available and thus no real notion of software development process existed (Avison & Fitzgerald, 2003). The focus was clearly on solving technical problems through programming and these programs were quite often embedded in the hardware (or even comprised a hardware solution).

Software development was the domain of the computer scientist and very few programmers had the organisational knowledge to understand the business for which they were developing software. These early programmers also often lacked the skills and training to perform efficient user requirement elicitation.

Small IT departments and limited resources also meant that the priority of programmers was to maintain existing systems, rather than to develop new software. This trend intensified as these informal or *ad hoc* systems were slowly integrated into business processes and became essential to the functioning of companies. Since this software was never designed as be mission critical, reliable, scaleable systems, a lot of energy was spent trying to ensure that they stayed online⁷.

Despite all these problems the demand for computerised business systems grew steadily. Companies came to realise that many business processes were much easier, quicker and more reliable when using computers. This demand increased to such an extent that most organisations came to rely on computer systems for the support of their core business activities.

This increasing reliance on information systems also meant a corresponding increase in the complexity (and cost) of the software being developed. In response companies put new emphasis on the analysis and design phase of software projects in order to try and improve planning and system features before programming started. Teams of programmers also had to start working together on the same software project, necessitating more formal development methodologies.

⁷See Boehm (2006) for a very thorough discussion of the difficulties faced by programmers and companies.

Table 2.1: *David Parnas’s list of current software problems*

Software is rarely delivered on time, schedules slip repeatedly.
The ultimate slip — cancellation, with nothing to show for the effort.
Software so poor that the system cannot be used.
System solves the wrong problem (requirements not met).
System out of date before it is in use (requirements change).
“Too many frills, not enough lifting power.”
Staff burn-out (software not maintainable by new staff).
Seemingly small changes cause huge effort.

Early software projects were not very good at coping with complexity (or even simply complication⁸) and could not deal with the millions of lines of code being produced. Increasing complication in software code and the cost of software exceeding that of hardware lead to widespread concern in the information systems arena, prompting several investigations into what is referred to as *the software crisis*.

In 1968 and 1969 the North Atlantic Treaty Organization (NATO) discussed this “crisis” at two conferences. Several issues were addressed including the “increasing size and complexity of software systems, difficulties in estimating costs and managing large numbers of people, shortages of skilled software professionals, emphasis on coding at the expense of design and testing, and poor documentation” (Lycett et al., 2003).

At this same conference the notion of *software engineering* was introduced by F.L. Bauer (Randell & Naur, 1968) and this really marked the beginning of a formalised, rational systems approach to software development. The standardised methodology was aimed at reducing costs and increasing productivity - addressing exactly those concerns raised by the “software crisis”.

Although some would argue that the software crisis was simply a phase or impetus in the development of a more robust software development regime, still others argue that we are still faced by the same issues today. David Parnas (Ågerfalk & Fitzgerald, 2006) argues that no *crisis* can persist for 40 years and this it is a chronic problem without any clear

⁸See Cilliers (2000) for a discussion on the very important distinction between complex and complicated systems.

cut answers. Some of the problems he identified are listed in Table 2.1.

Parnas’s contention is that these are perennial problems that the software community first tried to solve using the tools that they had at their disposal (algebra, logic, et cetera)⁹.

2.3.2 Early-methodology era

In an attempt to try and improve the quality of software systems and their development process, several companies started constructing more formalised models for software development. During the 1970’s organisations started identifying phases and stages of software development and brought these steps into a recognisable discipline. This approach was grounded in classical engineering and incorporated several “checkpoints” to ensure that each step had been successfully completed before proceeding to the next. This staggered approach was generally referred to as the Systems Development Life Cycle (SDLC) or “waterfall model” (Avison & Fitzgerald, 2003).

The underlying assumption for the SDLC is that requirements can be specified and defined. This approach is built on three principles¹⁰:

- *Formalism*—seeking a universal approach to a wide range of problem situations. An abstract set of processes, activities, and so on transform inputs into outputs to deduce a repeatable solution.
- *External knowledge capture*—capturing and standardising the development process to support a learning paradigm of inert knowledge transfer as a basis for coordination, communication, and training.
- *Economics*—using the division of process and labor to rationalise cost and effort. By establishing a purposeful framework of component activities, management can eliminate the activities that appear to be redundant, irrational, and counterproductive. It can also partition the process, allowing task specialisation and the paying of rates differentiated for particular skills.

The development of the SDLC has important parallels with project management devel-

⁹He goes further to argue that agile development is nothing new and provides no panacea to address these issues, but that discussion will follow later.

¹⁰Quoted from Lycett et al. (2003) (p. 81)

opment as detailed by (Koskela & Howell, 2002, 7-9)¹¹. The SDLC, however, also assumes that “thinking” can be isolated from “doing”. This separation is echoed in the planning model that requires all system design to be completed by the systems analyst before the “coding” is done by the software programmer. Computer programming is transformed from creative problem solving to simplified translation.

Many different variants of the SDLC still exist today with the most famous being that proposed by the National Computing Centre in the late 1960’s. Since this approach was the foundation for further methodologies, the main attributes are listed here:

1. The development process was divided into **phases** with each phase having to undergo testing and quality assurance before the programmers could progress to the next step. All the important steps and output (or “deliverables”) for each phase were clearly documented and could be verified by the project manager. These phases differed depending on the context, but often included feasibility, status quo analysis, business systems options, requirement definition, technical options, logical design and finally physical design. Some versions also include review and maintenance as formal phases.
2. **Techniques** were developed to aid in the completion of projects. These often included document templates (flow charts, organisational charts, document specifications, etc) and were meant to aid in the cost and benefit analyses of various different options.
3. Systems analysts were also aided by certain **tools**. These tools were normally software packages provided to aid with the development process. These toolkits were very broad and included everything from document specification to project management and diagram drawing.
4. **Training** courses were presented to analysts to help them adapt to the demands of their new jobs. In some cases these training courses terminated in a qualification for the attendees.
5. Lastly a **philosophy** was entrenched in the entire process. This could most often be seen as some form of “computer systems are usually good solutions to organisa-

¹¹Koskela & Howell (2002) deal with the concept *Theory of Management* and divide it up into the *Theory of Planning*, *Theory of Execution* and *Theory of Control*. These concepts relate to the SDLC, as it is intrinsically part of the project management process of any software project utilising this methodology.

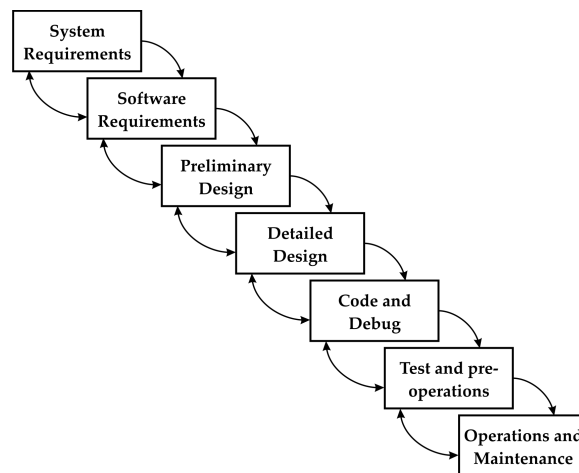


Figure 2.1: *The Classic Representation of the Waterfall Model*

SOURCE: Royce (1970)

tional problems and processing”. This embedded belief structure was pivotal for the rationalisation of the development attempt.

The SDLC (or *Waterfall model* (see figure 2.1) as it was called by Royce (1970) in his seminal paper *Managing the Development of Large Software Systems*) marked the move from informal towards more formal information system development¹².

A more formal, structured approach such as the SDLC certainly has some advantages. This approach has been deployed in many companies. Not only is the SDLC itself very well documented, but a whole library of case studies and documented implementations is also available. This focus on clear documentation enables good communication between different developers. The well-defined nature of the approach was also attractive to planners and managers leading to this approach being adopted by many large organisations. The rationale and *Weltanschauung* behind the waterfall model matched rational thinking in organisational theory at that time (Parnas & Clements, 1985). An intense focus on planning and design was the result of organisations trying to exercise control over the projects they were running by implementing a development methodology (Gasson, 1999).

¹²However it should be noted that a careful reading of Royce will show his model is indeed iterative and differs substantially from the classic interpretation of the waterfall model. His paper on this is unfortunately more often referenced than read. A further discussion of this is provided towards the end of this chapter.

2.3.3 Methodology era

During the 1980's there was a tremendous increase in the use of computers and information systems in organisations. This era also saw the start of interconnected networks of computers and the beginning of enterprise systems. The growth in the use of computer systems meant a corresponding increase in the number of software development projects being undertaken.

By now the limitations of the waterfall model had drawn serious criticism and companies were investigating alternatives to the structured formality of the SDLC. Many organisations were drawn to alternative approaches.

In general methodology development followed one of two routes: either it was developed by industry practitioners frustrated with the lack of support for their daily programming tasks or it came from a theoretical or academic investigation.

Most methodologies were simply a result of programmers trying to find some kind of “best practice” for software development. Most often these were simple “cook books” or practice notes combined into a somewhat more coherent set.

The majority of these attempts resulted in *techniques* rather than theoretically sound methodologies. The authors of these techniques were focused on sharing their own experiences and styles with others to try and improve the software engineering community. Many organisations in turn developed their own in-house methodologies for development. In companies where this was not seen as important, the programmers often left to work for competitors as independent consultants. A complicating factor was that the consulting services could be sold, but not the underlying “product” (the methodology) itself.

During this period the focus shifted to implementing some kind of modelling or flow-charts to help with the planning and design process (Aspray et al., 2007). This also opened the door to several facilitation tools and the importance of requirement elicitation became obvious.

Companies employing more effective development methodologies found that they had an important advantage over their competitors and could deploy more reliable systems in less time and for less money. This led to development methodologies gaining a strategic component and being recognised as a possible competitive advantage. (By now computers were fully integrated into many central business systems.)

This real-world use also meant that several integrative or synthetic approaches were

developed, eventually leading to what are now known as *blended methodologies* (Avison & Fitzgerald, 2003).

On the other side several universities and academic institutions were now studying software development as an academic interest. Computers and their use had become commonplace and software engineering was finally becoming accepted as a legitimate engineering pursuit if not a fully recognised discipline (Shaw, 2002).

Most of the academic investigations into software development methodologies tended to be theoretical and drew on knowledge from other disciplines and spheres (mathematics, engineering, business management, organisational theory, etc.). A few researchers did, however, start employing academic methods such as action research in the development of novel methodologies. These would later become relatively well-known models such as Soft Systems Methodology (SSM) and Multiview.

A study conducted in the United Kingdom by Fitzgerald (1999) found that most companies (57 per cent) claimed that they used some kind of formal methodology for software systems development, but only 11 per cent were using any (commercial) methodology in an unmodified form. This puts some perspective to the notion of “ready-made” methodologies. A full 30 per cent adapted some existing methodology to their own needs and 59 per cent indicated that they had developed their own, completely custom methodology (although they most often did take certain aspects from existing commercial methodologies).

As we can see, almost no companies surveyed used a formal methodology as-is. In almost all cases the approach is customised to the individual needs of the organisation and adapted to suit their environment. The study also casts some doubt on the true use and impact of formal commercial methodologies.

By this point it was patently clear that the waterfall model and appeal to rational classical engineering would not be the panacea the software industry had hoped for. Several articles and authors explored this and suggested that more flexible alternatives had to be found. Prominent articles include “Life-Cycle Concept Considered Harmful” by McCracken & Jackson (1982) expanding Dijkstra’s (1968) classic text “Go To Statement Considered Harmful”.

One of the main contenders among new methodologies was *Iterative Incremental Development* (IID), which has been developing since the 1930’s quality improvement cycles known as “plan-do-study-act” (PDSA) (Shewhart, 1939). From these roots it has undergone

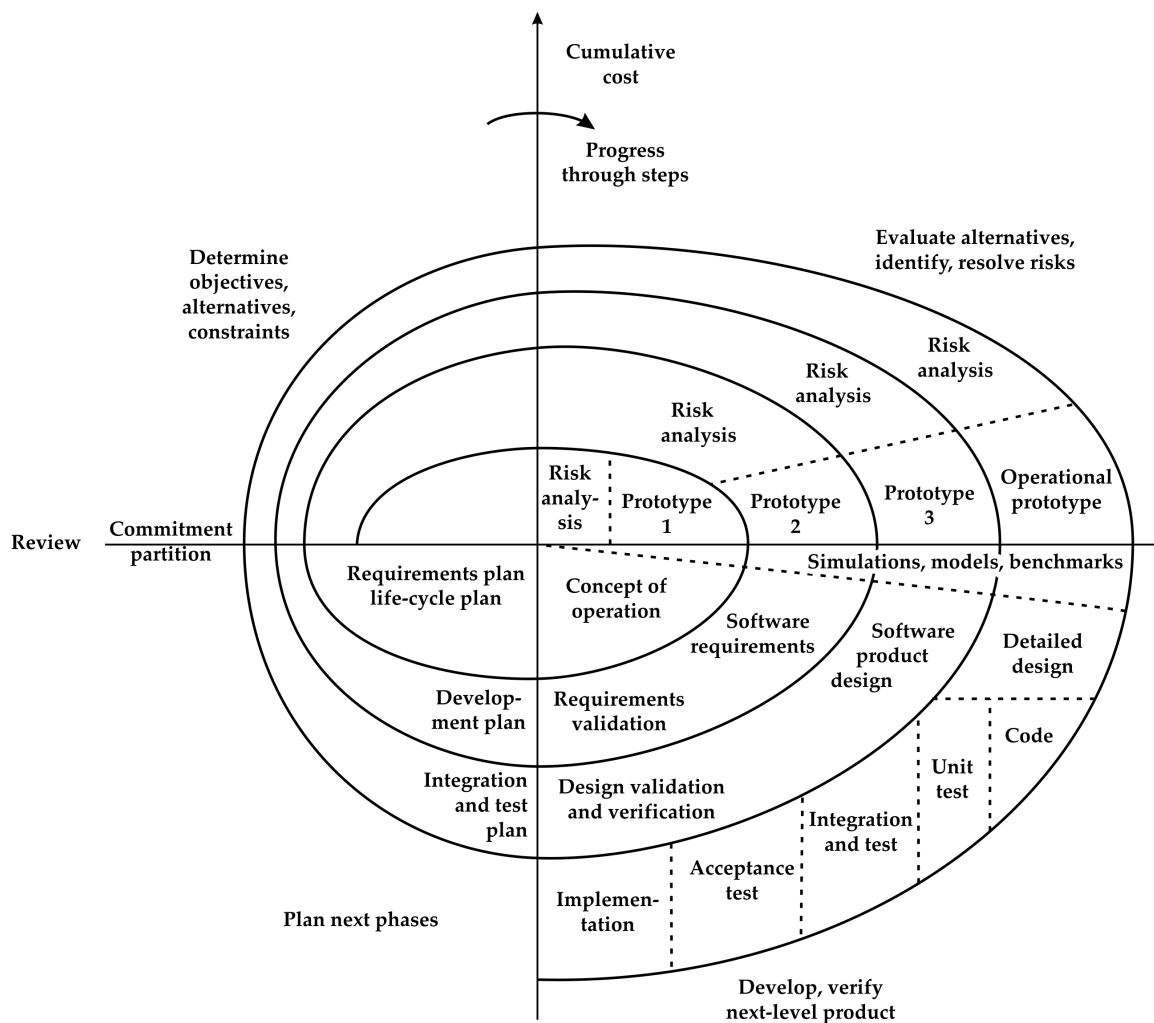


Figure 2.2: *The Spiral Model*

SOURCE: Boehm (1986)

several changes¹³.

One of the best known incarnations of IID is the *Spiral Model* promoted by Barry Boehm (1986). This spiral model attempts to integrate a top-down and a bottom-up approach to systems development. Planning is still a vital component in designing the system (and is initiated well before programming starts). After this planning stage a prototype design is created and through a variety of *spiralling* steps refined into the final product required.

This process also calls for thorough reviews at the completion of each of these stages. This review process is conducted by all the primary stakeholders.

¹³See Larman & Basili (2003) for an extensive historical analysis of the development of IID.

This reflexive, iterative, incremental process allows the development process to have some degree of sensitivity to changing demands and a changing environment. It also addresses one of the main criticisms against the waterfall model - that of being stuck with a linear, one-way understanding of the development process.

At the same time, the waterfall model was still very popular in rigid (high-risk) organisations such as the military, NASA and government departments (Wong, 1984) — although one should be fair and note that the waterfall model was mostly applied in a modified form integrating elements of IID (Larman & Basili, 2003).

The mainstream acceptance of IID meant that criticism against the waterfall model was now well-known and companies had to actively consider their internal processes for software development. In “No Silver Bullet” Brooks (1987) shares a certain understanding that the world has become more complex and unpredictable than traditional processes allowed for:

Much of present-day software acquisition procedure rests upon the assumption that one can specify a satisfactory system in advance, get bids for its construction, have it built, and install it. I think this assumption is fundamentally wrong, and that many software acquisition problems spring from that fallacy.

This clearly spoke to the difficult reality that real-world software developers were faced with. The rapidly globalising world, which impacted the environment in which companies were operating, also started to have a profound impact on the people building systems to support these business processes.

Fitzgerald (1996) suggests that companies also faced additional pressure to standardise their methodologies to comply with the requirements set by certain standards bodies. He lists the ISO (International Standards Organisation) and the SEI (Software Engineering Institute) as particularly important in this regard.

2.3.4 Era of methodology reassessment

The next phase in the historical development of software methodologies took place in late 1980's and most of the 1990's. Although there was the potentially exciting prospect of IID implementation in the early 1980's, many organisations had already become disillusioned with the idea of a formal software methodology - especially due to the failings of most of these methodologies in practice.

Software programmers were finding that not only the waterfall model, but the entire notion of software methodology was not the panacea they had hoped for. Many of the original problems experienced in the early days of software programming were still equally prevalent with a methodology as without.

Some organisations responded by switching between different methodologies in an attempt to find a suitable approach. Still others completely abandoned all structured approaches to software development (Avison & Fitzgerald, 2003).

Although there were certainly problems with many of the methodologies in use, one should also note that many organisations implemented these methodologies for the wrong reasons (or implemented them badly). Some organisations tried methodologies to formalise processes, others to increase control or accountability. With a varied set of uses and requirements, satisfying these expectations would have been an impossible task for any methodology - especially since most of them were not developed for this purpose.

Avison & Fitzgerald (2003, p. 583) give a very good overview of this “backlash against methodologies”. A short summary is presented here to allow the reader to follow the argument. The main criticisms are generic and do not relate to a single methodology and include:

- *Productivity* — Methodologies do not really reduce development time and add many new forms of overhead. Deploying a specific methodology is resource intensive and slow.
- *Complexity* — Many approaches try to address all concerns and as a result become very difficult to implement due to the level of detail required.
- *‘Gilding the lily’* — Methodologies develop requirements to a level that is not required in practice. These thorough requirement elicitations often end in ‘wish lists’ rather than real requirements.
- *Skills* — Significant skills are required to implement these methodologies.
- *Tools* — Required tools are often very difficult to use and do not generate enough benefits to justify the effort. Focus shifts to documentation rather than the product being developed.

- *Not contingent* — The methodologies do not take into consideration the size and complexity of the product being developed and thus result in a large amount of overhead for small products.
- *One-dimensional approach* — Normally one way is selected in which to address the development of a project. This can result in underlying issues or problems being ignored.
- *Inflexible* — Methodologies do not allow for changes to requirements during development. Modern companies operate in an environment filled with constant change and this can lead to the development of an irrelevant product.
- *Invalid or impractical assumptions* — Most methodologies need to make certain fundamental assumptions (such as a stable environment). In practice many of these assumptions cannot be supported.
- *Goal displacement* — Often strict adherence to the methodology becomes more important than the development of a good quality product. Wastell (1996) discusses the limiting effect of methodologies on creativity.
- *Insufficient focus on social and contextual issues* — Methodologies focus on technical problems and their solutions, without the necessary consideration of the social context in which the final system needs to function. Hirschheim et al. (1996) argues that this narrow focus should be expanded to cater for systems development that is emergent, historically contingent, socially situated and politically loaded.
- *Difficulties in adopting a methodology* — Adopting a methodology is hard in practice. Organisations which are used to building new software systems without such a structured approach, often find it hard to adapt.
- *No improvements* — Most importantly it is often argued that the implementation of these methodologies (as indicated here, often a difficult task) has not led to better systems.

A detailed analysis of these criticisms is outside the scope of this thesis, but one should note that some of these could as easily be attributed to inadequate methodology selection or poor implementation as to faults in the design of the methodologies themselves.

In this era of methodology re-assessment, organisations had a few different paths to explore¹⁴.

The most obvious (but not necessarily most advantageous) option would be to jettison methodologies altogether and return to some form of ad hoc development. Although this extreme route does save some resources it does nothing to address the concerns that lead to the development of methodologies in the first place. This approach is termed *amethodical systems development* by Truex et al. (2000).

An alternative, logical, option is to continue the search and development process for better software methodologies. Developments in terms of object oriented programming (discussed later) were especially instrumental in the continuing evolution of methodologies.

Another option was for organisations to deploy a variety of methodologies depending on the context of the specific product they were developing. While this did address some of the concerns, it also meant that the standardised process promoted by picking a unified methodology was lost.

Lastly several organisations opted to simply get rid of the problem by employing external development. Software development was either outsourced or packaged solutions were bought and deployed in the organisation. Tailorable packages also became available and could be rolled out as ERPs (Enterprise Resource Packages). While this might have addressed some of the issues for companies themselves by shifting the responsibility to the vendors, it did not solve anything for software development as a discipline.

During this era of re-assessment in the early 1990's the internet started taking a more prominent position and all programmers had to start dealing with the reality of massive interconnected information systems. The notion of a "fixed environment" was finally put to rest and all systems had to cope with constant dynamic change.

This tumultuous environment lead to the introduction of several relatively radical IID approaches. Grouped together, they became known as *lightweight methodologies*. These new methodologies also, once again, borrowed from traditional or classical engineering and especially from developments such as *lean manufacturing* and *lightweight design*.

By the 1990's there was a shift towards modularised design and specifically *object-oriented programming* (OOP). At the same time the discipline of *software architecture* was

¹⁴Once again Avison & Fitzgerald (2003) provides a good, detailed overview which is presented in an adapted form here.

also starting to mature and for the first time positioned itself between technology and process (Royce & Royce, 1991)¹⁵.

A myriad of more flexible approaches were starting to appear. These all have a basis (to some extent) in IID, but have a wide variety of distinguishing features (and heritage). A short summary¹⁶ is provided in Table 2.2.

This enormous new set of methodologies brought about some serious questions about the quality implications of following these new approaches. To address these, several new standards (such as ISO 9000 and CMM) were introduced (Lycett et al., 2003). Many of these had the philosophy that one should “say what you do, then do what you say you do”. These considerations are especially important due to agile methodologies specifically focussing on the quality of the product that you produce as a key factor in the development process.

The *Capability Maturity Model* (CMM) was introduced by the Software Engineering Institute at Carnegie Mellon University in the early 1990’s to try and assist in a more objective evaluation of software development quality. This model has been largely superseded by the newer *Capability Maturity Model Integration* (CMMI)¹⁷. The newer formulation incorporates sensitivity to modern iterative development techniques and can be used in the evaluation of projects employing IID or agile processes.

At this stage several institutions were busy trying to develop a more formalised approach and create some order in this fast developing field. These include the Institute for Electrical and Electronic Engineers (IEEE), Association for Computer Machinery (ACM) and other major players. Cooperation such as this led to the development of the *Software Engineering Body of Knowledge* (SWEBOK). This document aims to consolidate the generally accepted knowledge required by software engineers and practitioners in related disciplines.

¹⁵This notion and the development of software architecture as an essential component of the development process is discussed in detail by Kruchten et al. (2006)

¹⁶The table is adapted and expanded from work by Dyba & Dingsoyr (2008). This comparison is discussed in the next chapter.

¹⁷This model is still being maintained by SEI and is currently in version 1.2

Table 2.2: *Description of main agile development methods, with key references*

Agile Method	Description	Reference
Adaptive Software Development (ASD)	An adaptation of RAD assuming constant adaptation to be a normal process. Introduces speculate (consider possible interpretations and stakeholder errors during planning), collaborate (balance work between predictable and uncertain factors) and learn (short iterations of development followed by introspective analyses) cycles.	Highsmith (2000)
Crystal methodologies	A family of methods for co-located teams of different sizes and criticality: Clear, Yellow, Orange, Red, Blue. The most agile method, Crystal Clear, focuses on communication in small teams developing software that is not life-critical. Clear development has seven characteristics: frequent delivery, reflective improvement, osmotic communication, personal safety, focus, easy access to expert users, and requirements for the technical environment	Cockburn (2004)
Dynamic software development method (DSDM)	Divides projects in three phases: pre-project, project life-cycle, and post project. Nine principles underlie DSDM: user involvement, empowering the project team, frequent delivery, addressing current business needs, iterative and incremental development, allowing for reversing changes, high-level scope being fixed before project starts, testing throughout the lifecycle, and efficient and effective communication	Stapleton (2003)
Extreme programming (XP; XP2)	Focuses on best practice for development. Consists of twelve practices: the planning game, small releases, metaphor, simple design, testing, refactoring, pair programming, collective ownership, continuous integration, 40-h week, on-site customers, and coding standards. The revised XP2 consists of the following primary practices: sit together, whole team, informative workspace, energised work, pair-programming, stories, weekly cycle, quarterly cycle, slack, 10-minute build, continuous integration, test-first programming, and incremental design. There are also 11 corollary practices	Beck (2000), Beck (2004)
Feature-driven development (FDD)	Combines model-driven and agile development with emphasis on initial object model, division of work in features, and iterative design for each feature. Claims to be suitable for the development of critical systems. An iteration of a feature consists of two phases: design and development	Palmer & Felsing (2002)
Lean software development (LSD)	An adaptation of principles from lean production and, in particular, the Toyota production system to software development. Consists of seven principles: eliminate waste, amplify learning, decide as late as possible, deliver as fast as possible, empower the team, build integrity, and see the whole	Poppendieck & Poppendieck (2003)
Open Source Software Development (OSSD)	The development of a model exposing system source code to the public. The participatory approach aids with the improvement of software quality and transparency. Especially efficient in using a community approach to bug resolution.	Raymond (1999) (et al)
Rational Unified Process (RUP)	Adaptable process framework developed by IBM for use in the software development process. Includes breaking projects up into building blocks and lifecycle phases for better planning. Augmented by six engineering disciplines and three supporting disciplines. Six best practices are also defined to minimise faults and increase productivity.	Kruchten (2004)
Scrum	Focuses on project management in situations where it is difficult to plan ahead, with mechanisms for empirical process control; where feedback loops constitute the core element. Software is developed by a self-organizing team in increments (called sprints), starting with planning and ending with a review. Features to be implemented in the system are registered in a backlog. Then, the product owner decides which backlog items should be developed in the following sprint. Team members coordinate their work in a daily stand-up meeting. One team member, the scrum master, is in charge of solving problems that stop the team from working effectively	Schwaber (2004)

2.3.5 Age of agility

Despite several valiant attempts throughout the 1990's and 2000's, the dissatisfaction with software methodologies grew. Nandhakumar & Avison (1999) argue that traditional methodologies “are treated primarily as a necessary fiction to present an image of control or to provide a symbolic status”.

The effect of globalisation has not only meant a change in the operations of businesses, but has also impacted on the software development process itself. This has meant that both the finished software product and the development thereof need to be able to cope with the new stresses brought about by a rapidly changing world. This quite often means that software is developed by several different teams in different parts of the world. Herbsleb & Moitra (2001) provide us with a summary of several new demands that methodologies are being forced to cope with. When looking at agile methodologies, it is important to bear in mind that they also have to deal with these new challenges that are a reality of software development in the current global environment.

By the early 2000's the progress in software development methodologies culminated in a group of proponents of so-called “lightweight” methodologies discussing the future development of this development approach¹⁸. As a result of discussions (and even a “Lightweight Methods Summit” in 2001) a *Manifesto for Agile Software Development* was compiled.

Although this brought about a certain degree of organisation in the movement, the term *agile development* encompasses several methodologies (as indicated in Table 2.2) and there are several varying understandings of this term. For the purposes of this study a more coherent working definition is considered in the next chapter.

¹⁸“Culminate” here refers to the historical development of agile methodologies, rather than the entire software practice as such. Agile development is not necessarily the logical progression of methodology evolution and this study in no way argues that the history (and future) of software methodology development follows a single path or narrative. Many authors would in fact argue that agile development is nothing new (Ågerfalk & Fitzgerald, 2006) or does not exist at all. This issue receives further discussion in a later chapter.

Chapter 3

Agile Development

The previous chapter has outlined, in some detail, the historical roots of contemporary software development methodologies leading up to the introduction of agile development. The lack of agreement on a formal definition for this practice (Abrahamsson et al., 2002) necessitates a closer look at the characteristics of agile development in general.

Additionally some specific methodologies are selected (from the vast array of options) for the analysis presented in this thesis.

3.1 Establishment of the ‘Agile Development Movement’

The disillusionment with structured formal planning methodologies has lead to more flexible or *agile* approaches gaining popularity. Several of these approaches (or in the parlance of the authors concerned “processes”) developed independently.

The term “agile” here is taken from the common meaning¹ - “*having the faculty of quick motion; nimble, active, ready*” and is intended to reflect the new level of adaptability required by an age of increasing turbulence.

In 2000 Kent Beck convened a conference in Oregon to discuss what was then referred to as “Light” or “Lightweight” methodologies (Highsmith, 2001)². Although these novel approaches to software development had evolved independently, it became clear that there

¹Taken from the Oxford English Dictionary (1989).

²Highsmith (2001) notes that Alistair Cockburn identified there was general disgruntlement with the term ‘Light’: “I don’t mind the methodology being called light in weight, but I’m not sure I want to be referred to as a lightweight attending a lightweight methodologists meeting. It somehow sounds like a bunch of skinny, feeble-minded lightweight people trying to remember what day it is.”

Table 3.1: *Twelve principles of agile development*

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.
Deliver working software frequently, from a couple of weeks to a couple of months, with a preference for the shorter timescale.
Business people and developers must work together daily throughout the project.
Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
Working software is the primary measure of progress.
Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
Continuous attention to technical excellence and good design enhances agility.
Simplicity – the art of maximising the amount of work not done – is essential.
The best architectures, requirements, and designs emerge from self-organising teams.
At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

were several common aspects and that it could be beneficial to discuss them.

After this initial meeting a follow-up was held in Utah in February 2001. At this meeting the participants decided to use the term *agile* as an umbrella to refer to these newer methodologies standing in contrast to “documentation driven, heavyweight software development processes” (Highsmith, 2001). The importance of this meeting was the adoption of the *Manifesto for Agile Software Development* agreed to by all the participants³. This manifesto states four central tenets (Beck et al., 2001) (These four core tenets are expanded by the twelve ‘principles’ listed in Table 3.1.):

- *Individuals and interactions* over processes and tools
- *Working software* over comprehensive documentation
- *Customer collaboration* over contract negotiation
- *Responding to change* over following a plan

³These included representatives from Extreme Programming, SCRUM, DSDM, Adaptive Software Development, Crystal, Feature-Driven Development, Pragmatic Programming and others.

In each item the proposal on the left is favoured over the traditional idea on the right.

The participants to this conference later formed a non-profit organisation called *The Agile Alliance* with the purpose of promoting and advancing agile development processes. Several companies also offer “certification courses” aimed at formalising training for agile development processes⁴.

By aligning many different “alternative” approaches and iterative practices, the group could also develop their image and branding and thus market this approach in the software development community. A coordinated front was also required to give legitimacy to an approach that would be involved in large-scale projects. Many traditional company executives would not be comfortable trusting their development processes to a “group of independent thinkers”. The formalisation of the alliance could quell some of these concerns.

The Agile Alliance currently has more than 4 000 members and is also responsible for the organisation of an annual conference and several smaller “programmes” (AgileAlliance, 2010b). The goal of this alliance is not to design a single unified methodology, but rather to coordinate the myriad of independent methodologies adhering to the agreed set of tenets and principles accepted by the members of the alliance.

3.2 Background

As noted in earlier chapters, there is intense disagreement on the exact definition and delimitation of the concept “agile development”. Indeed many authors⁵ have argued that agile development is not really a separate type of development methodology, but rather a natural evolution of existing practices. Many academic debates have grappled with this issue, including the well-known feature articles in Cutter IT Journal - The Great Methodologies Debate: Part 1 and Part 2⁶.

It is important to bear in mind that the use of agile development processes does not preclude any other methodologies. Many organisations successfully apply a wide range of different development styles. In fact Hawrysh & Ruprecht (2000), indicate that a single

⁴The Agile Alliance has a position paper on the matter discussing the relative merits of different types of certification. The Alliance itself does not provide or endorse any certification. (AgileAlliance, 2010a).

⁵Listed and argued at length by Ågerfalk & Fitzgerald (2006).

⁶Cutter IT Journal, November (Vol. 14, No. 11), December 2001 (Vol. 14, No. 12) and January 2002 (Vol. 15, No. 1)

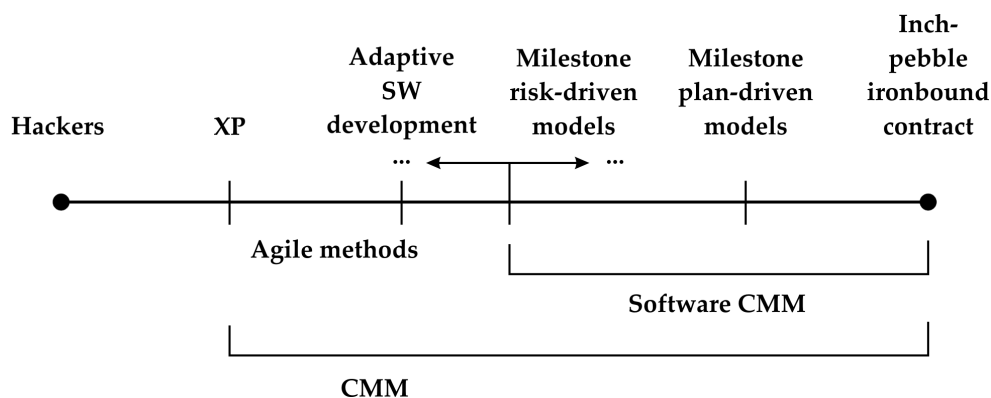


Figure 3.1: *The Planning Spectrum*

SOURCE: Boehm (2002, p. 65)

methodology is likely to fail and advise that project managers should carefully consider the specific project before selecting a development approach⁷.

Boehm (2002, p. 65) argues that all development (especially plan-based methodologies) are located on a spectrum of increasing emphasis on plan (reproduced in Figure 3.1). Agile methods are of course more unstructured than micromanaged milestone planning (in his terms “*inch-pebble ironbound contract planning*”), but still more rigid than disorganised hacking. Also note the scope of the CMM with regards to the planning spectrum.

3.2.1 Central themes

The “tenets” listed in the agile manifesto can be further expanded as central themes relevant to all approaches grouped under the umbrella term *agile*.

Individuals and interactions

The first important shift is the move away from a central planning process completed in advance. Key decisions are now made by the developers and other team members as they become important. This shift not only means greater communality between developers, but also improved ownership of the project by ordinary developers.

A golden thread running through agile development discussions is the newfound emphasis being placed on the human factor involved in software development. In fact Cockburn & Highsmith (2001), go as far as to say “agile development teams focus on individual com-

⁷This view is shared by many other authors, including McCauley (2001); Glass (2001).

petency as a critical factor in project success”. This brings about a shift in the conceptualisation of the “software programmer”. The amount of decision making (and concomitantly responsibility) devolved to the individual developer is greatly increased.

Highsmith & Cockburn (2001, p. 121) comment that “what is new about agile methods is not the practices they use, but their recognition of people as the primary drivers of project success, coupled with an intense focus on effectiveness and manoeuvrability. This yields a new combination of values and principles that define an agile world view.”

This is an important break with the conception of the “worker” promoted in scientific or behaviourist management. Gone is the notion that the worker is a generic “factor of production” that can be replaced or interchanged whenever necessary (Taylor, 1919).

It also breaks with neo-classical economics and idea of *homo economicus*. Human beings, workers, team members are now recognised as idiosyncratic individuals with their own skills, ideas, strengths and weaknesses. Team structure and functioning are adapted to harness these differences, rather than attempting to mould individual workers to fit in pre-defined job descriptions.

While this certainly conforms to the distributed management model in use in agile projects, it does also pose certain challenges. For example, Constantine (2001, p. 68) identifies a problem: “There are only so many Kent Becks in the world to lead the team. All of agile methods put a premium on having premium people...”. Boehm (2002) poignantly adds to this by noting the statistical dilemma of 49.99999 per cent of programmers being below average⁸.

The quality (in terms of skills, ability and tacit knowledge) of the individuals involved in the agile development project is largely responsible for the adaptability achieved by electing these development approaches. Moving decision making from some prepared plan to an individual, presupposes that the individual will be able to handle the added responsibilities. In practice, this does mean that some exceptional (or at least above-average) developers are required to effectively run an agile project.

⁸Whether this statistical observation is relevant and sensible or just a rhetorical device is left to the reader.

Working software

The second important distinction of agile development is the (infamous) reduction in documentation vis-à-vis traditional approaches. The guiding principle is that working code is more valuable than planning documents. To cope with quality concerns, agile development introduces the notion of continual testing of all completed modules. Highsmith & Cockburn (2001, p. 121) lists “the unforgiving honesty of working code” as one of the basic concepts of agile development:

Working code tells the developers and sponsors what they really have in front of them — as opposed to promises as to what they *will* have in front of then.

The working code can be shipped, modified, or scrapped, but it is always *real*.

An additional benefit of avoiding copious documentation is the promotion of simplicity. For example, one of the traps avoided by agile development is referred to as YAGNI (“You Aren’t Going to Need It”) (Boehm, 2002)⁹.

To compensate for the lack of formal documentation, agile development once again relies on the human developers involved in the project. The close and frequent interaction between members, brought about by the organisational structure and processes of an agile team, fills the gaps left by the lack of documentation. This is also augmented by the strong focus on continual testing and refactoring, enabling future developers to understand the source code directly, rather than having to rely on external documentation.

Additionally proponents of agile development employ the concept of “self-documenting code”. By using meaningful variable names, classes and structures, the functioning of the code becomes apparent without the need to resort to formal documentation.

Customer collaboration

Agile development gives precedence to customer collaboration over contract negotiation. This does not mean that contracts are no longer needed. On the contrary, the importance of the contract increases commensurately with the size of the project (Abrahamsson et al.,

⁹Here Boehm, however, notes that this approach only works well when you *assume* an unstable and changing environment. In relatively stable environments it may indeed be quite sensible and responsible to do the planning and architecture of new systems in advance. This fundamental assumption of uncertainty becomes a contentious issue within the field of software development in general.

2002). The shift is that the contract negotiation session itself becomes part of building a relationship with the customer.

By directly involving the customer in the development process, many changes and adaptations can be effected as soon as they become apparent. This constant interaction (combined with rigorous procedures for limiting changes during a development session) assists the developers in creating products that are as close to customer expectations as possible.

Responding to change

The final (and possibly best known) theme found in agile development implementations is the deep concern with adaptability to change. Instead of the change management processes followed in traditional development methodologies, development team members and customer representatives are empowered to constantly adapt the project to keep up with changing real-world requirements.

Boehm's life cycle cost differentials theory explains that the cost of change increases throughout the project's development life cycle. To minimise the cost of producing a relevant and useful product, it therefore become essential that necessary changes are made as soon as they become apparent (Highsmith & Cockburn, 2001). Agile development enables these quick changes by devolving decision making authority to the developers, rather than requiring higher level approval.

At the core of this responsiveness to change, is the iterative development paradigm (Favare, 2002). By accepting change and improvement and fundamental presuppositions of any development process, it becomes easier to adapt without falling into the trap of shifting targets.

This sensitivity to change not only allows for greater adaptability, but also shortens the life-cycle of the project. Miller (2001) explains this compression by referring to nine characteristics of this development approach:

1. Modularity on development process level
2. Iterative with short cycles enabling fast verifications and corrections
3. Time-bound with iteration cycles from one to six weeks
4. Parsimony in development process removes all unnecessary activities

5. Adaptive with possible emergent new risks
6. Incremental process approach that allows functioning application building in small steps
7. Convergent (and incremental) approach minimises the risks
8. People-oriented, i.e. agile processes favour people over processes and technology
9. Collaborative and communicative working style

3.3 Summative Description

Although this thesis is concerned with the spectrum of phenomena linked to agile development, it is helpful to have a concise description as a reference point. To this end the definition provided by Abrahamsson et al. (2002, p. 17) is particularly helpful and is used as a working summative description for this study:

What makes a development method an agile one? This is the case when software development is *incremental* (small software releases, with rapid cycles), *cooperative* (customer and developers working constantly together with close communication), *straightforward* (the method itself is easy to learn and to modify, well documented), and *adaptive* (able to make last moment changes).

3.4 Prevalence of Agile Development

Since the inception of agile methodologies in the early 2000's, the use of these methodologies in companies has steadily grown and has no doubt become a viable alternative or supplement to traditional development practice.

Exact figures on the adoption rate of agile processes differ according to the study conducted. Scott Ambler (2008) from the IBM Rational programme conducted several adoption rate studies and found that between 2006 and 2009 65-69% of respondents indicated that their companies were employing (or also employing) agile approaches¹⁰. This is echoed

¹⁰Read his comments for a discussion on the discrepancy between developer reported figures and management reported figures.

by VersionOne (2009) who also found 65-84% adoption rate. More studies by Forrester Research (West et al., 2010) indicate that 63% of organisations surveyed used agile development methodologies with 35% relying on it exclusively.

Regarding the specific implementation, all found SCRUM to be the most prevalent, attaining more than 50% presence in all studies. This is closely followed by a SCRUM/XP hybrid and then several smaller methodologies in different orders depending on the survey.

Due to the popularity of Scrum and Extreme Programming (XP) this study will use these two approaches as a demonstration of the implementation of the typical agile development process. Since they also operate on different levels of abstraction it provides a reasonably representative view of this type of ‘methodology’.

3.5 Scrum

As reported in the previous section, Scrum¹¹ is the agile development methodology with the highest reported use in companies today. This is especially remarkable given that it is one of the newer methodologies to have been developed.

Sutherland & Schwaber (2007) credit Takeuchi & Nonaka (1986) as the inspiration for calling this development approach ‘Scrum’. In their article Takeuchi and Nonaka describe adaptive, quick, self-organising product development in Japan and compare it to the game of rugby. The metaphor refers to the gameplay in rugby where the players have to devise a strategy to get “an out-of-play ball back into the game” by using teamwork¹².

The approach was developed by Jeff Sutherland at Easel Corporation in 1993 (Sutherland, 2004). During this period he closely cooperated with Kent Beck leading to the concurrent development of XP and Scrum as complementary development processes. Sutherland & Schwaber (2007, p. 11) quite bravely state “agile development is now accepted globally as the best way to develop, maintain and support software systems” and continue to argue in detail why Scrum can be seen as a replacement methodology applicable both to small

¹¹The name of this approach was originally ‘SCRUM’ (all capitals) as used in the article formalising this methodology (Schwaber, 1995), but in later work the convention is changed to a mixed case label ‘Scrum’. There is no semantic difference between the two names.

¹²Sutherland & Schwaber (2007) also cite several other authors as an inspiration and influence for the Scrum approach. These include lean development (Poppendieck, 2005), Japanese development (Liker, 2004; Holford & Ebrahimi, 2007), knowledge management (Takeuchi & Nonaka, 2004) and Senge (1990).

scale and enterprise-level development¹³.

In promoting the Scrum process, Sutherland & Schwaber (2007, p. 12) boast that it will “add energy, focus, clarity, and transparency to project planning and implementation”. They also lists some of the main benefits, boldly saying it will:

- Increase speed of development
- Align individual and corporate objectives
- Create a culture driven by performance
- Support shareholder value creation
- Achieve stable and consistent communication of performance at all levels
- Enhance individual development and quality of life

Scrum in essence attempts to guide team members in their software development process with specific reference to the constantly changing the environment. The approach does not prescribe specific software development techniques (Abrahamsson et al., 2002).

Schwaber (1995) discusses how the system development process is complicated and complex. This results in inevitable changes in the environmental and technical variables of a project during the development process. Scrum tries to accept this challenge by modifying the development process to be sensitive to change and quick to adapt. Schwaber (1995, p. 8) reiterates:

Evolution favours those that (sic) operate with maximum exposure to environmental change and have optimised for flexible adaptation to change. Evolution deselects those who have insulated themselves from environmental change and have minimised chaos and complexity in their environment.

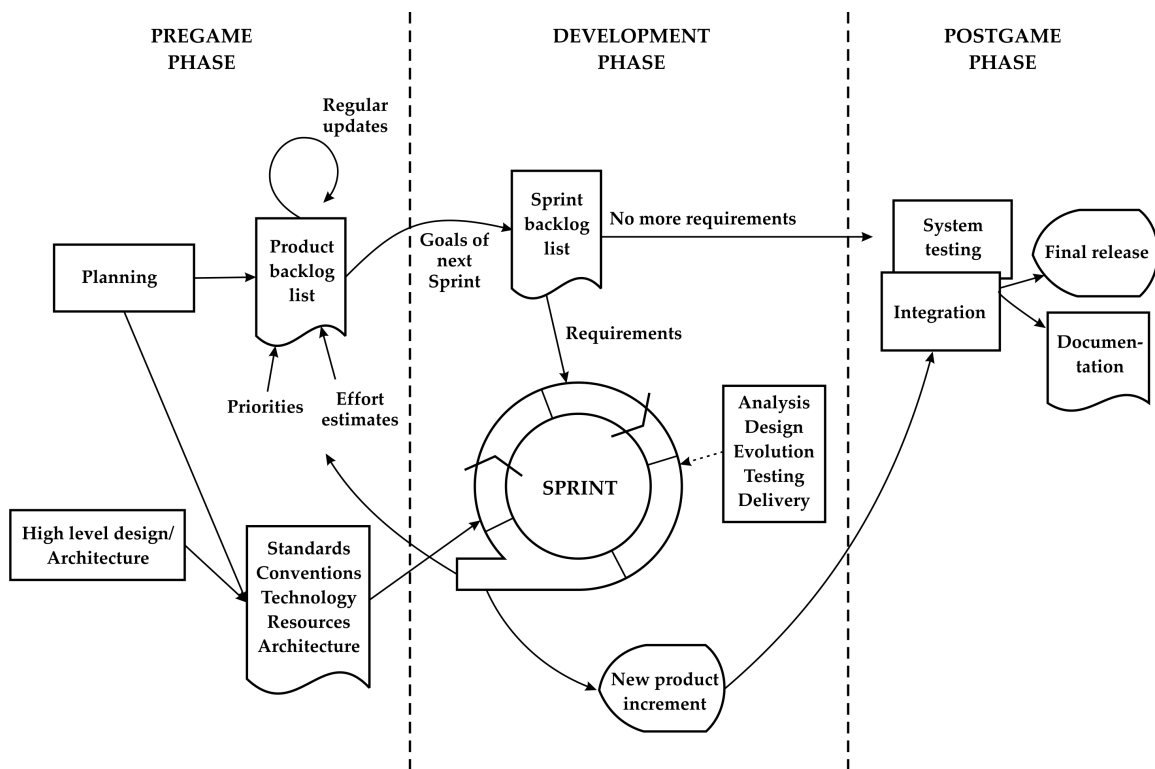


Figure 3.2: *The Scrum Development Process*

SOURCE: Abrahamsson et al. (2002, p. 28)

3.5.1 Process

The Scrum methodology groups activities into three phases: *pregame*, *game* and *postgame*¹⁴. A diagrammatic representation by Abrahamsson et al. (2002) is reproduced in Figure 3.2.

Pregame

The pregame phase starts off with a *planning* step. This is a relatively linear process aimed at reaching a definition for the system under development. The requirements for the finalised product are elicited from all stakeholders (customers, sales, support, marketing,

¹³Once again the opinions quoted here are clearly as much an attempt at marketing as it is a comment on the efficacy of the approach. When reading work by the “core group” of agile writers, this agenda cannot be ignored.

¹⁴The following section describes Scrum and its process, roles, artefacts and ceremonies in some detail and is based on several descriptions, including Schwaber (1995); Abrahamsson et al. (2002); Schwaber (2004); Schwaber & Beedle (2002); Norton (2007); Deemer & Benefield (2007); Sutherland (2007). The terminology is particular to the world of Scrum and it is presented without commenting on the peculiarities of this vocabulary and phraseology.

development, etc.).

A list of requirements is compiled and specific items are prioritised. This list is called the *product backlog*. After the compilation of this list, individual requirements get assigned an estimated time and cost implication. This phase deals with both conceptualisation and analysis.

Although this step is formalised and defined, new requirements are constantly added to the backlog and the priorities adjusted. This phase also deals with the establishment of the project team, internal processes, tools, documentation rules and testing procedures. This entire planning process is revisited after every development iteration (sprint).

The planning step is followed by a *architecture* step. This step aims to address high-level design issues and specify the implementation of items on the backlog.

Game

The actual product development happens during the game phase. A development cycle is generally 30 days or less and are called a *sprint*. A *sprint backlog* is produced from the product backlog. This list contains all the requirements to be addressed in a single sprint.

The development process is continually adapted to the variables of time, requirements, quality, cost, resources, implementation technologies and tools and competition. This gives Scrum development much greater agility than would be provided by traditional approaches.

The sprint starts with a planning meeting during which the team commits to specific items. At the start of each working day the team meets for a short while to report on progress.

At the end of the sprint all new code is released to production (no extensions allowed). This ensures that development is done in smaller increments and allows for extensive testing of the code produced during each sprint. All items developed during a sprint should be ready to ship by the last day of that sprint.

Postgame

At the end of the project the product backlog is compared to the delivered code to ensure that all requirements have been met. A *closure* phase is then started to verify final testing, prepare final documentation and release the product.

3.5.2 Roles

Depending on the source consulted, Scrum has between three and six defined roles. Roles in this sense refers to actual human actors involved in the product development process. At a minimum the roles of *Product Owner*, *ScrumMaster* and *Team* are identified. Several sources augment these by including the *Customer*, *User* and *Manager*.

Product Owner

The final responsibility for the project resides with the Product Owner. This individual has final decision making power over the composition and prioritisation of the tasks on the Product Backlog. He/she should also take all the inputs about the definition of the product (including those by the customer, team members and other stakeholders) into consideration and define the product vision. The Product Owner may in some cases be the customer, but this is not necessarily the case.

Additionally the Product Owner has the responsibility of determining release dates and altering feature requirements and priorities with every sprint. The owner also decides whether the work of the team is acceptable and is in charge of running sprint planning meetings.

It is important to note that although the Product Owner has several responsibilities and should be available to the team at all times, the owner does not instruct the team about specific tasks since they are self-organising and self-managing.

ScrumMaster

The ScrumMaster is responsible for “doing whatever is necessary to help the team be successful” (Sutherland & Schwaber, 2007, p. 23). The idea is for the ScrumMaster to support the team by helping with facilitation and removing obstacles to the development process. The ScrumMaster also has to help shield the team from outside interferences and ensure close cooperation and coordination between the different roles and functions.

The ScrumMaster also looks after interpersonal relationships and ensures that communication is frequent and efficient. The ScrumMaster is responsible for running the daily scrum meetings as well as the sprint planning and review meetings. In this role the master facilitates discussions and monitors the process to update the burn-down chart and other artefacts. The role of ScrumMaster is a supporting role - not a managing one.

Generally the division of roles means that it is not recommended for the ScrumMaster and Product Owner to be the same person.

Team

A scrum team is cross-functional and includes all persons involved in the development of the product. The ideal scrum is composed of five to ten persons. All participants needed for actual development should be included in the team (for example designers, programmers, testers, researchers, etc.). If more than 10 to 15 people need to be working on a single product, the team should be split into several smaller teams and functionality should be divided between these different scrums.

In general it is not recommended that members split their time between several projects. Personal commitment is needed to ensure the required level of buy-in from all team members. At the beginning of each sprint team members select tasks and commit to finishing them within that sprint.

The team is totally responsible for their own management and organisation. This is an organic process not mediated by any kind of appointed leader.

Customer

Although the customers are not directly involved in the development process, they are consulted in determining the Product Backlog and understanding the requirements of the system under development. In many cases the customer and user roles overlap.

Manager

Managers in the process interact with the scrum team and set the requirements, standards and conventions needed in the development process. They also help in the product definition and requirement elicitation and are in charge of final decision making.

In the scrum process, managers do not in fact *manage* the team, but rather act as mentors or experts coaching the team. This often means serious changes need to be made to managers' existing management styles.

3.5.3 Artefacts

As part of the scrum process several documents and tools (called *artefacts*) are produced.

Product Backlog

The product backlog contains all the features and requirements of the product to be addressed in the entire project. These items are also prioritised based on the value they offer to the customer. Features on this list are accompanied by a description of the technical prerequisites or implications involved in including them.

Each item on the product backlog also has an estimated implementation time assigned to it. Items are units of work small enough to be completed in a single sprint. As bugs, defects are identified and enhancements requested by the customer, these are also added to the product backlog. At any point during the project the product backlog can be used as a single definitive view of “everything that needs to be done”.

The level of detail required for the product backlog is determined by the Product Owner and team. Items close to the top of the list (and thus close to development) will be more detailed. As items move up the list, the Product Owner has the responsibility to further describe and specify them. The entire product backlog is the responsibility of the Product Owner.

Release Backlog

The release backlog is similar to the product backlog, but contains only the items that are to be integrated in the current release. Additional information may be added in this backlog to enable the team to develop these features.

Sprint Backlog

Each sprint is started by the compilation of a sprint backlog. This list is compiled by the Product Owner based on the priorities on the product backlog.

Items are selected from the sprint backlog by the team members (not assigned to) in consultation with the Product Owner and ScrumMaster. Once the sprint backlog has been compiled it cannot be changed until the end of the sprint. As soon as the sprint is complete (the time period elapsed or all items have been finished) a new sprint backlog is compiled for the next sprint.

All items on the sprint backlog should require less than 16 hours of work to complete. Any tasks longer than this are broken up into two or more tasks.

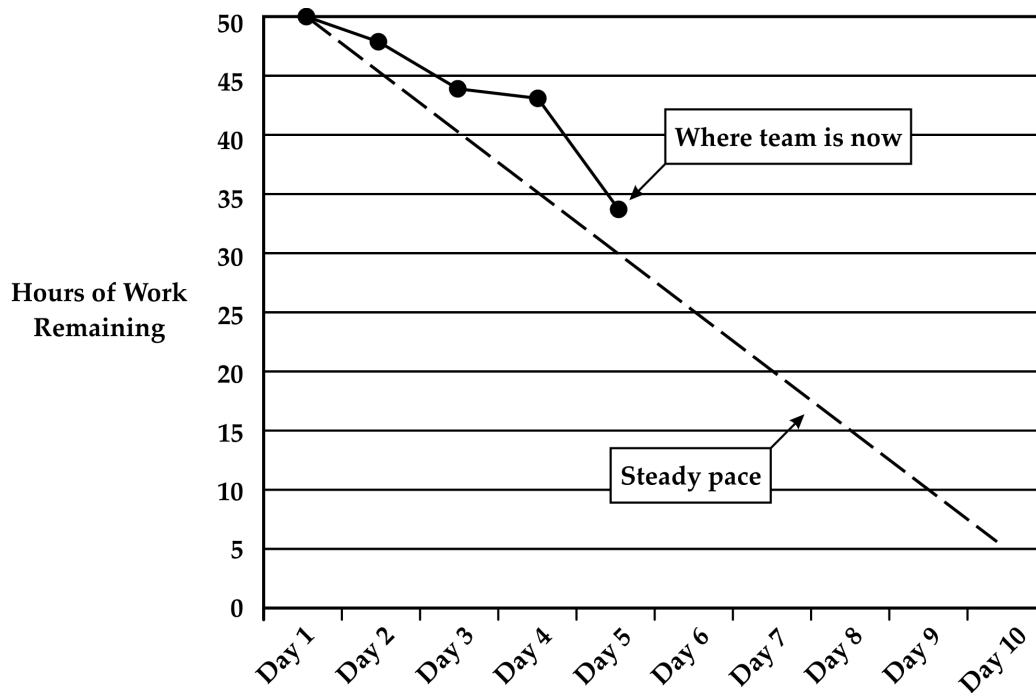


Figure 3.3: *A Burndown Chart*

Burndown Chart

This chart is one of the visual tools used in the Scrum process (an example is reproduced as Figure 3.3.). It is a visual representation of the amount of work remaining. This very practical tool gives all members the ability to instantly judge whether they are on track to deliver the tasks in the current sprint by the end of the period.

The remaining tasks (expressed in hours) are represented on the Y-axis with the remaining time shown on the X-axis.

Several versions of the burndown chart may be produced, depending on the requirements of the specific scrum. These may include a sprint burndown chart and a product burndown chart.

The chart is updated daily and ideally it should slope down so that the line on the chart reaches zero by the end of the time period. Once again, Scrum takes a different approach from normal project management, by displaying only the actual work completed (expressed as tasks with a certain amount of hours assigned to them), not the amount of hours spent on the project.

By glancing at the burndown chart, members of the team can easily discover if their current pace will allow them to complete the sprint successfully.

3.5.4 Ceremonies

Meetings are referred to as “ceremonies” in the Scrum process. Although formal meetings are kept to a minimum, any Scrum process will have at least a *Sprint Planning Meeting*, *Daily Scrum Meetings*, a *Sprint Review Session* and a *Sprint Retrospective Session*.

Sprint Planning Meeting

The first step towards initiating a new sprint is the sprint planning meeting. As at all meetings, the ScrumMaster is responsible for facilitating this session. The first step in this meeting is for the Product Owner and Team to review the Product Backlog and consider all of the tasks currently listed.

Once the general product backlog has been discussed, members proceed to select items from this list and commit to implementing them in the current sprint. These items are then transferred to the sprint backlog. Team members get to decide which tasks they commit to, rather than having tasks assigned to them.

This meeting generally lasts a few hours and should not be rushed since decisions made here have serious implications for the rest of the sprint. At this meeting team members adjust the estimates of all the tasks they commit to. They also give an indication of the time available per day to work on the sprint. (This is the total working hours per day minus time spent on administration, maintenance, meetings and other commitments.)

Once the team has finalised their tasks for the current sprint, the Product Owner cannot change or add to it. In cases where the external circumstances significantly change, the Product Owner can request that the entire sprint be abandoned. This is a very serious step and means all team members have to stop their work and restart with a new planning meeting.

The idea behind this rigidity is to protect the team from changing goals and external interference in their development work. Once committed to, team members can be sure that their tasks will not change during the sprint giving them some stability in the development environment. This also aids in managing the power balance between the Product Owner and the Team, forcing the Owner to carefully consider the priorities of tasks on the Product Backlog.

Daily Scrum Meeting

During a sprint a meeting is held daily. It lasts no longer than 15 minutes and always takes place in the same venue. The ScrumMaster is in charge of this meeting and the whole Team should attend the meeting¹⁵. The idea is to quickly determine the progress of the project and to consider any issues that may be hampering development.

Each member of the team is asked to simply answer three questions:

- What have I achieved since yesterday?
- What will I achieve before the next meeting?
- What's stopping me from achieving what I want to achieve?

Using the answers from these questions, the ScrumMaster can work to try and eliminate any obstacles to the process. The information provided will also help the Master to update the burndown chart and draw attention to open tasks. These frequent meetings will also allow the ScrumMaster to become aware of any lingering inter-personal issues needing attention.

Sprint Review Session

At the end of each sprint all code developed during that sprint is demonstrated to the Product Owner (and often customers, users and managers). This is an informal meeting that is timeboxed to a maximum of four hours (although it may take as little as 10 minutes). The idea is that this is a simple demo, not an extensive presentation.

All participants get the opportunity to give feedback and help make decisions regarding the next development sprint.

At this meeting the Product Owner also makes the decision about which items in the current sprint have been satisfactorily completed.

¹⁵Individuals not part of the team may attend these meetings, but they are not allowed to talk, ask questions or influence the meeting in any way. The distinction is often explained with a short parable:

“The reasoning behind this is often explained with the story of a pig and a chicken planning to open a restaurant. The pig asks ‘What shall we call it?’ to which the chicken replies ‘Ham and eggs’. ‘No thanks,’ says the pig. ‘You’re involved, but I’m committed’. ”

The point is that only the team is truly committed to the development of the product and as such should not be influenced by other individuals - at least not during this part of the process.

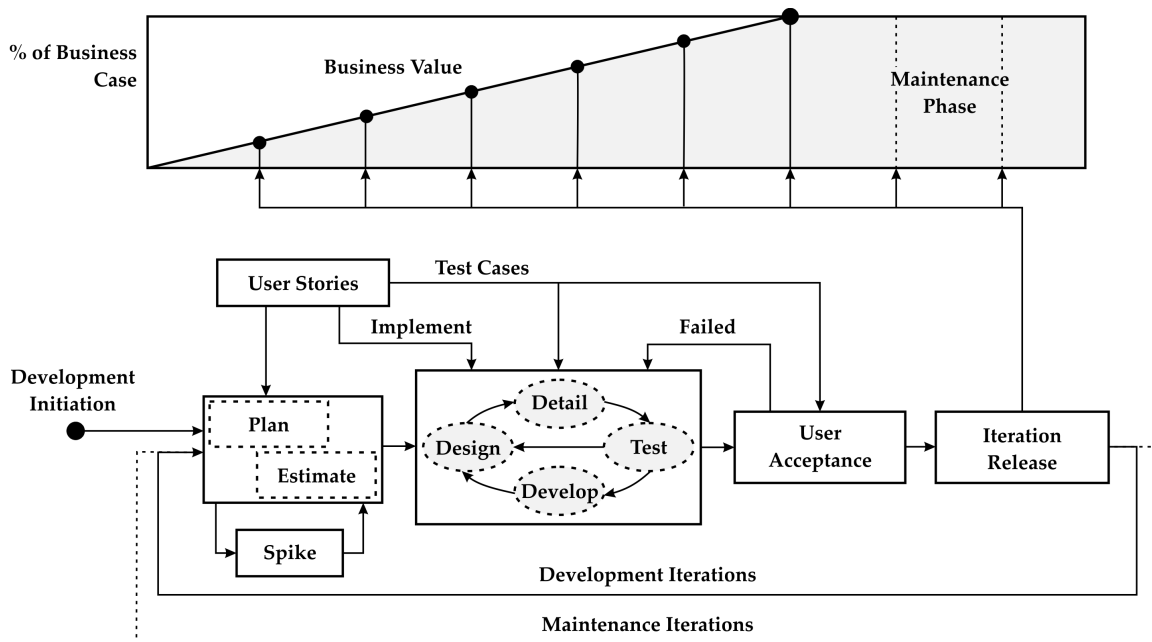


Figure 3.4: *XP Life-Cycle (Traditional View)*

SOURCE: Norton & Murphy (2007, p. 3)

Sprint Retrospective Session

After the Review Meeting, the ScrumMaster sets up a Retrospective Meeting during which members discuss and review what went well during the sprint. This meeting is also aimed at identifying improvements in the Scrum process that can be implemented during the next sprint. This session lasts no more than three hours and is normally facilitated by a neutral, external person (often a ScrumMaster from a different team).

Topics for discussion include questions such as “What is working well” and “What can be done better”.

3.6 eXtreme Programming

Extreme Programming is distinct from Scrum in that it does not offer a set of prescriptive procedures. The core notion of XP is to instil certain core *values*, *principles* and *practices* in the development process and allowing the development team to define their own processes accordingly (Norton & Murphy, 2007). In contrast and as complement to Scrum a short

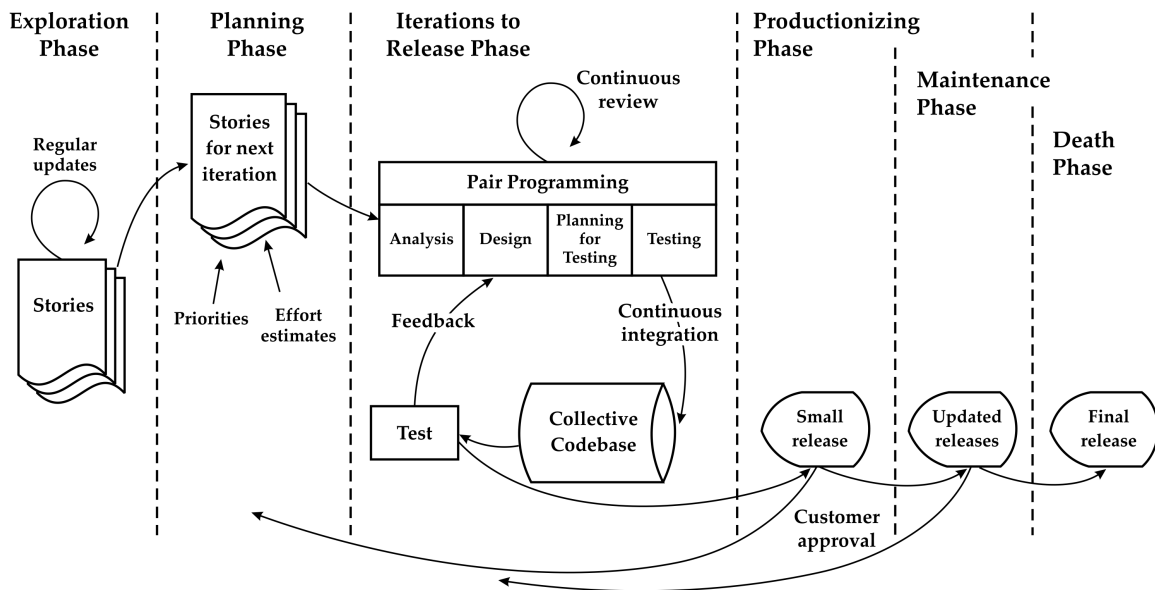


Figure 3.5: *XP Life-Cycle (Expanded View)*

SOURCE: Abrahamsson et al. (2002, p. 19)

overview of XP will follow¹⁶.

XP developed based on the experiences of Kent Beck (and others) during the C3 project at Chrysler. This methodology was developed as an evolution of the serious problems experienced due to the long development cycles of traditional development approaches (Abrahamsson et al., 2002)¹⁷.

3.6.1 Process

Although XP does not set out a defined list of tasks and sequence of steps it does divide the life-cycle of the project into six phases: *Exploration*, *Planning*, *Iterations to Release*, *Productionizing*, *Maintenance* and *Death*. The entire XP-methodology is outlined in Figure 3.4, with Figure 3.5 breaking it up into the appropriate phases.

¹⁶This section is based on descriptions in Abrahamsson et al. (2002); Beck (2000, 2004); Cao et al. (2004); Haungs (2001); Norton & Murphy (2007); Beck & Andres (2000); Stotts et al. (2003).

¹⁷In fact, Abrahamsson et al. (2002) account that Haungs (2001) described it starting as ‘simply an opportunity to get the job done’.

Exploration phase

The XP process starts with all the participants writing *story cards* outlining their requirements for the first release of the product. This phase lasts a few weeks to months with participants discussing the technology and architecture to be used in the project. The scope of the project is negotiated (called a *negotiated scope contract* (Beck & Andres, 2000)) contrasting with the traditional approach to project planning.

Planning phase

As in Scrum, the next step is the estimation of the development time needed to complete each of the requirements noted on story cards. After the estimation is done, the stories are prioritised and divided between different programmers. Normally a development cycle should not exceed two months.

Iterations to release phase

The actual development of the product will include several iterations before it is ready to be put into production. Each iteration should take between one and four weeks to complete. The customer is involved in selecting the stories appropriate to each iteration.

Productionizing phase

After the developers feel the product is ready for release a set of extensive testing and checking procedures is initiated. Any changes required at this stage should be carefully considered for implementation.

Maintenance phase

The maintenance phase deals with all development requests subsequent to the product being released into production. The majority of these tasks are generated by the customer service department. The team responsible for the product during this phase does not need to be the same as the team who did the initial development.

Death phase

As soon as the customer is content with the product development and no further development is required, the project enters the Death Phase. During this phase definitive documentation is prepared and no more changes are made to the code. The project can also be sent into the Death Phase if further development becomes too expensive or the product is deemed irrelevant.

3.6.2 Values, Principles and Practices

The crux of XP is that the approach is seen as a value-, principle- and practice-driven methodology. Beck (2004, p. 14) makes a distinction between *values*¹⁸ as a high-level driver (“Values are the roots of the things we like and don’t like in a situation.”) and *practices* as the “evidence of values”¹⁹.

To bridge this gap between *values* and *practices*, Beck (2004, p. 15) introduces the concept *principles*. These “domain-specific guidelines for life” are expanded into normative formative influences on the development process.

Values

The guiding *values* identified as being conducive to successful development include:

- *Communication*: A common vocabulary is required for efficient team functioning. Communication is also used to create a sense of team and effective cooperation.
- *Simplicity*: The development team should aim for the most elegant and gracefully simple solutions to programming challenges. This ensures that the code remains understandable and eliminates errors introduced by over-complicated coding. This strive for simplicity should be context sensitive.
- *Feedback*: Reflexivity is promoted in the system by receiving frequent feedback from all stakeholders (including the customer). This feedback is augmented by systemic

¹⁸Please note that the use of the term “values” is common practice in the agile community when used in the specific case mentioned here, but it should not be confused with the social sciences understanding of values.

¹⁹Although several objections can be made to the definitions Beck assigns to these terms, the distinction is the important point being made here.

Table 3.2: *Principles of Extreme Programming*

Humanity:	This involves the people who need support and recognition. Individuals need work and personal life balance.
Economics:	The software and development process has to have business value.
Mutual Benefit:	The solution should benefit the whole, rather than individuals or only part of the team.
Self-Similarity:	Teams should look for repeating patterns during the design, testing and project life cycle.
Improvement:	Individuals and teams should do the best they can today and strive for improvement tomorrow.
Diversity:	Teams need a mix of skills, experience and personalities. This creates greater flexibility and a high-quality product.
Reflection:	Individuals and teams should have formal and informal opportunities to reflect on the project.
Flow:	Projects should have a continuous rhythm of delivering value. They should not adopt a stage gate approach.
Opportunity:	Teams should see problems that arise as opportunities to develop new strengths and remove weakness.
Redundancy:	XP practices approach problems of design, testing and other activities from multiple angles.
Failure:	Although failure isn't something to be sought, it's often unavoidable when exploring solutions and building knowledge.
Quality:	Schedules and budgets should not be maintained at the cost of quality. A project can progress at a substantial pace, while maintaining process and product integrity.
Baby Steps:	Organisations and teams need to accept that adopting new behaviours should be done one step at a time. Too much change is counterproductive.
Accepted Responsibility:	Individuals and teams explicitly accept responsibility, which is accompanied by authority and control of related tasks.

feedback and testing (both unit testing and user acceptance testing).

- *Courage:* This value refers to the moral fibre required by the team to be willing to change and discard completed code and work with constant changes to their requirements and specifications. This also means team members have to commit to building the right solution even if it is more effort than settling for the easy one.
- *Respect:* The underlying value enabling the enactment of all the others is respect. Team members should accept and promote the contributions of each team member.

Principles

Norton & Murphy (2007, p. 5) provide a good summary of the 14 principles providing the interface between values and practices. This summary is reproduced in Table 3.2.

Practices

In the first formulation by Beck (2000), 12 practices are listed as being core to XP. This list is, however, expanded in the later edition of *Extreme Programming: Explained* (Beck, 2004) and refined to 24. This list is divided into core practices (forming the framework) and advanced practices for more mature teams. These more advanced practices should also be tried gradually as the team becomes more confident.

Norton & Murphy (2007, p. 6-7) provide us with a synoptic list of all these practices as reproduced in Table 3.3.

Roles

Within the XP process, roles are not intended to be fixed and rigid. Beck (2004) identifies several roles including *Testers*, *Interaction Designers*, *Architects*, *Project Managers*, *Product Managers*, *Executives*, *Technical Writers*, *Users*, *Programmers* and *Human Resources*. The idea, however, is not that these individuals function only within the space of their own roles, but contribute as full members of the team.

Table 3.3: *Practices of Extreme Programming*

CORE PRACTICES
Sit Together: Teams should share a common space to promote collaboration and effective communication.
Whole Team: Have the necessary skills and experience on hand. Adopt the principles of cross-functional teams.
Informative Work Space: Use the working environment to reflect project status and captured knowledge.
Energised Work: Individuals should not be working excessive hours. Balance productivity with sustainable project demands.
Pair Programming: Coding should be done by two programmers sitting at one computer to produce better designs, testing and code.
Stories: Use a visible mechanism to capture user requirements with a short narrative. Estimate project time based on the user stories.
Weekly Cycle: Adopt an iterative approach based on a weekly cycle of planning and delivery.
Quarterly Cycle: Use quarterly intervals for longer-term planning around a theme. Ensure continued business alignment.
Slack: Build contingency into the weekly and quarterly cycles. Avoid overcommitting and underachieving.
Ten-Minute Build: Strive to make the build cycle as fast and easy as possible to encourage up-to-date builds.
Continuous Integration: Changes in the code should be integrated into the system as soon as the code is written.
Test-First Programming: Develop test cases upfront. Automate where possible, and integrate into test harness.
Incremental Design: Avoid big upfront design. Let the design emerge in an evolutionary fashion cycle-by-cycle.

ADVANCED PRACTICES
Real Customer Involvement: This goes beyond the whole team practice to include active and sustained customer involvement.
Incremental Deployment: The team should demonstrate value to the business by deploying functionality increments throughout the project.
Team Continuity: Minimising changes to the team increases collaboration effectiveness and yields a sense of stability.
Shrinking Teams: As teams grow in capability and experience, they should look to reallocate resources when those resources are no longer needed.
Root-Cause Analysis: Teams should build integrity into products and processes by removing defects and the causes of defects.
Shared Code: Once a team has developed a sense of shared responsibility, code ownership should move from the individual to the team.
Code and Tests: These are the primary artefacts for development and maintenance. The team should remove waste in other project artefacts.
Single Code Base: There should be a single code stream to prevent parallel versioning and configuration.
Daily Deployment: Each daily build is deployed to the business. This requires a mature testing and deployment process and environment.
Negotiated Scope Contract: Fixed cost, quality and time should be balanced against variable scope and delivered features.
Pay per Use: This involves adopting a funding model based on the actual use of the software. This provides cost and quantitative feedback.

Chapter 4

Sensemaking

This chapter conceptualises *sensemaking*¹ as a theory with which to analyse the organisational activities referred to as ‘agile development’. This theory will be applied in an attempt to develop insight into underlying cognitive process and nature of agile development, before engaging with the effects these new development methodologies have on the organisation and its structure. Sensemaking will be used to fill gaps in our understanding of organisational theory and problematise the notion that agile development is simply a new “recipe” for “building software”.

4.1 Conceptualising Sensemaking

As with all interpretive functions, defining theories becomes challenging and problematic given the rejection of platonic absolutes. This is also true in the case of sensemaking where a concise definition remains elusive. The term ‘sensemaking’ is used in many fields, ranging from cognitive psychology to information systems, communication theory and organisational science. In this research the main focus is placed on the work done by Karl Weick (organisational sensemaking) with some additional inspiration from Brenda Dervin (sense-making and communication) and other authors.

Weick et al. (2005, p. 409) quote Taylor & Van Every (2000, p. 275) when they start their treatment of sensemaking as a concept: “[S]ensemaking is a way station on the road

¹Karl Weick labels the theory ‘Sensemaking’, Brenda Dervin refers to her version as ‘Sense-making’ and some other authors use the term ‘Sense Making’. Since this treatment will deal predominantly with Weickian theory the term ‘sensemaking’ will be used when referring to the concept described in this chapter. Where important the relevant specific theory and style will be used.

to a consensually constructed, coordinated system of action”.

In its most basic form, Weick (1993, p. 634) says the basic idea of sensemaking is “that reality is an ongoing accomplishment that emerges from efforts to create order and make retrospective sense of what occurs”. It literally means “the making of sense” (Weick, 1995, p. 4). By actively engaging with stimuli, actors come to grips with them and construct sense (or meaning). In effect they “structure the unknown” (Waterman, 1993, p. 41).

Dervin (1992, p. 2) defines sense-making as “a theoretic net, a set of assumptions and propositions, and a set of methods which have been developed to study the making of sense that people do in their everyday experiences ... In essence, then, the term *Sense-Making* refers to a coherent set of theoretically derived methods of studying human sense-making”.

The mechanism through which individuals² structure their world (and subsequently assign meaning) starts with the acknowledgement that a perceived reality does not match the expectations of that reality. This causes the actor to select certain stimuli, which is followed by an attempt to place these stimuli in some sensible framework (Weick, 1995, p. 4-5).

Weick (1995, p. 13) proposes a specific approach to applying hermeneutics in this context by making an important distinction between sensemaking and interpretation, arguing that the ongoing nature of the act of sensemaking is an essential characteristic thereof, whereas interpretation can be either an action or an artefact (or both). Interpretation here is defined as an attempt to deal with dissonance³ by appealing to some external reality, while sensemaking is posed as a rejection of this notion of objective reality and rather deals with the creation (or construction) of meaning in the ambiguous or uncertain situation. This brings us to the conclusion (Weick, 1995, p. 14) that to engage in sensemaking is to “construct, filter, frame, create facticity . . . , and render the subjective into something more tangible”.

Sensemaking is inherently hermeneutic and puts specific focus on the creative; only after a constructive process can the actor proceed to interpret stimuli, thus “making sense”. Sensemaking is therefore a process both of authoring and of interpretation.

²Take note that we are here referring to individual, human actors. The theory of sensemaking will later be applied to the organisation.

³Dissonance theory was an important precursor for the development of Sensemaking Theory (Weick, 1995).

4.1.1 Historical Roots

The purpose of this thesis is not to provide an extensive discussion of sensemaking, but rather to propose it as a theoretical framework with which to interpret agile development as an organisational practice. This discussion of sensemaking would, however, be remiss if it did not contain some brief mention of the historical roots and heritage from whence the theory developed.

Sensemaking was greatly influenced by *ethnomethodology* and specifically the work done by Garfinkel (1967, p. 104-115). These studies lead Weick to conclude that sensemaking contributes to maintaining and sustaining the structures which allow individuals to function, by moulding the *Lebenswelt*⁴.

The process of sensemaking (described later in more detail) developed from the field of *cognitive dissonance theory* (Festinger, 1957). This theory relates to the retrospective rationalisation individuals undertake to try and restore consistency in their mental models when confronted with *postdecision* dissonance. Quite often individuals will then mould their evaluation of decision options to skew it in favour of the chosen alternative. Weick (1995) postulates that the same behaviour is displayed when actors are faced with a shock in their experience of reality, thus prompting the act of sensemaking.

Although Weick (1995, 43) (in his seminal book) makes but one very brief reference to Dilthey and Heidegger⁵ it can be posited that sensemaking is *fundamentally* a hermeneutic activity. Sensemaking borrows from the ontological and phenomenological traditions in its attempt to deal with reality construction.

While there are many interesting issues regarding the nexus between sensemaking and philosophy, these (unfortunately) fall outside the boundaries of this study.

Dervin (1999, p. 42), however, acknowledges this heritage,

“Sense-Making starts with the fundamental assumption of the philosophical approach of *phenomenology* - that the actor is inherently involved in her observations, which must be understood from her perspectives and horizons.”

Her work on the topic focusses predominantly on sense-making in communication. Al-

⁴This realisation is amplified by the influences of social constructivism (Berger & Luckmann, 1967).

⁵Only as read in Burrell & Morgan (1979) and Winograd & Flores (1986) and only in reference to ‘thrownness’ and the ongoing nature of sensemaking.

though her work is relevant to certain aspects of this study, the main emphasis will be the conceptualisation of sensemaking as seen in the work of Karl Weick.

4.2 The Nature of Sensemaking

In order to apply sensemaking theory to agile development, a concise description of the theory first needs to be established. In this section attention is paid to the famous ‘seven properties’ of sensemaking identified by Weick. Since these properties apply to all instances of sensemaking (and in his application are focused on individual sensemaking) these properties are also expanded with several other themes and aspects needed to understand sensemaking in the organisational context.

4.2.1 Interpretation vs. Authoring

As previously stated, the nature of sensemaking is *inherently* hermeneutic. This easily leads the reader to believe that sensemaking is simply another modern/post-modern theory on the interpretation of texts. The focus on interpretation is prevalent in many (if not most) contemporary academic enquiries, but especially in the humanities, social and business sciences. The fundamental nature and the vast applicability thereof in the ‘coping’ with real world dilemmas give it great relevance.

Weick (1995), however, clearly makes the point that sensemaking is not (only) about interpretation. In fact his contention is that sensemaking is predominantly about authoring, rather than reading. He argues that the interpretive act in sensemaking is *necessarily* preceded by a creative one. Whereas interpretation concerns itself with the interpretation of a text, sensemaking is also about authoring that text. Weick also introduces a cognitive element to the sensemaking act, thereby rendering the activity a ubiquitous part of all human experience.

As with all interpretive acts, sensemaking focuses its attention on a specific *text*. This ‘text’ should, however, be conceptualised as the object being interpreted and can encompass the entirety of the actor’s reality; in the same sense as used by Derrida when he claims “*Il n’y a pas de hors-texte*” (“There is no outside-the-text”). The sensemaking act is not simply a philosophical or academic endeavour, but has profound ontological and existential implications. The seriousness of the sensemaking process is another way in which it can be

distinguished from interpretation.

For Weick sensemaking is a common act, continually being performed by many actors, which aims to take disparate sense perceptions and therewith construct a coherent text to be interpreted. The conceptual intricacies of this act of construction and interpretation is outside the scope of this study, save to expand briefly on the nature of the interpretation.

In the same text, Weick refers us to Mailloux (1990) for a definition of interpretation as “acceptable and approximating translation”. Equating interpretation to translation immediately presents us with three distinct elements: the text, the actor (audience) requiring the interpretation and the interpreter. Weick (1995, p. 7) conceptualises this interpretive act as being situated in a given community and introducing the complications of “political interests, consequences, coercion, persuasion and rhetoric”.

Intuitively the reference to interpretation as translation would presuppose some underlying “true meaning” as one finds in the mathematical communication model (Shannon, 1948). The interpretation act then simply becomes a transaction between the signifier and the signified. This would mean that sensemaking is simply an act of *discovery* and accurate mediation between the audience and some underlying “real-world” text⁶.

When confronted by a perceptual shock, leading to an opportunity for sensemaking, individuals often instinctively appeal to some platonic notion of the “text in the world”. This possibility is rejected by sensemaking. By discarding the idea of an external reality, actors are left with no option but to create (formulate) their reality before engaging in the interpretation thereof.

Rather than engaging in discovering the *acceptable* conceptualisation of the problem situation, sensemaking directs the actor to *construct* a relevant text and then proceed to give meaning to that construction. Of course this construction does not happen *de novo*, but requires selecting a variety of very real cues - thus making sensemaking an interwoven process of both discovery and creation (or if you like, authoring and interpretation)⁷. The implication is that sensemaking is as much about discovery as it is about *ignoring* certain cues and choosing not to act upon them (Weick, 2001, p. 460).

This generative act transforms the problem definition into an active process. The first

⁶Weick (1995) also briefly notes this problematic in reference to work done by Daft & Weick (1984).

⁷Weick et al. (2005, p. 409) expands on this, stating “Sensemaking is about the interplay of action and interpretation rather than the influence of evaluation on choice”.

step in making sense of a situation is defining exactly what that “situation” actually is. Problem setting is thus elevated from a passive step to an active one. The importance of “formulating the mess” is also apparent in some systems approaches such as Russel Ackoff’s *Interactive Planning* (Jackson, 2003)⁸.

Weick (1995, p. 13) summarises the difference between sensemaking and interpretation as interpretation being a “product”, while sensemaking is not only a product but also an “activity” or “process”.

Since sensemaking deals with reality construction and not merely interpretation, the effect of the sensemaking activity transcends mere “coping” or “comprehending”. As a result this activity has an impact on the actor’s entire perception and experience of the world/reality (or *Weltanschauung*).

4.2.2 The Sensemaking Activity

Although sensemaking is often referred to as an ongoing conversation (Weick, 1995, p. xi), it is possible to discern an *activity* of sensemaking. Understanding the processes that take place within this activity is paramount to grasping the mechanics of sensemaking in organisations.

Sensemaking is brought into focus when the actor is confronted by a specific situation that does not make sense. This *surprises* the actor and generates cognitive dissonance: “Explicit efforts at sensemaking tend to occur when the current state of the world is perceived to be different from the expected state of the world, or when there is no obvious way to engage the world.” (Weick et al., 2005, p. 409).

In other words, the normal stream of sense perceptions have become unintelligible, jolting the actor into a state of conscious unease. This unease is caused by a gap between the expected cues and those observed. The gap is construed by the actor to be a state of chaos.

Sensemaking is the act of bridging that gap and actively *ordering* the observed chaos (Weick et al., 2005, p. 411). This involves taking the perceived cues and attempting to place them into a frame (viz. a *frame of reference*) (Weick, 1995, p. 4).

Tsoukas & Chia (2002, p. 570) expand on the organising function of sensemaking:

⁸For a more detailed explanation of Interactive Planning, refer to Ackoff (1974)

“Organisation is an attempt to order the intrinsic flux of human action, to channel it toward certain ends, to give it a particular shape, through generalising and institutionalising particular meanings and rules.”

Gioia et al. (1994, p. 365) explicates this process by noting three sequential steps⁹:

1. Sensemaking occurs when a flow of organisational circumstances is turned into words and salient categories.
2. Organising itself is embodied in written and spoken texts.
3. Reading, writing, conversing, and editing are crucial actions that serve as the media through which the invisible hand of institutions shapes conduct.

4.2.3 Seven properties of Sensemaking

In his authoritative work on the process of sensemaking, Weick (1995) identifies seven key properties of sensemaking. He describes sensemaking as:

1. Grounded in identity construction
2. Retrospective
3. Enactive of sensible environments
4. Social
5. Ongoing
6. Focused on and by extracted cues
7. Driven by plausibility rather than accuracy

Weick proceeds to argue that these properties are *defining* elements of the process of sensemaking. In the Battered-Child Syndrome (BCS) case he refers to, he concludes that the activities displayed in that context can be classed as sensemaking due to their fit with the mentioned properties¹⁰.

⁹As described in (Weick et al., 2005)

¹⁰He states clearly (p. 3) “*Thus* BCS is an instance of sensemaking *because* it involves...” and lists the seven properties (emphasis added).

Given the obvious importance of these properties, they will be summarised in this section¹¹. It should, however, be noted that this is only the first step, establishing that the presence of these seven properties could constitute some process of sensemaking. This is not yet enough to argue that the sensemaking is anything more than an individual process and certainly does not move the observed activity into the realm of *organisational* sensemaking. This also becomes apparent in the title of Weick's book (*Sensemaking in Organizations*) versus other works discussed later dealing with the organisational component (*Making Sense of the Organization*). This argument is expanded later in this section.

These properties are often used as a type of check-list or framework with which to evaluate a given phenomenon and then analytically decide if the observed events can be described as an act of sensemaking. This is, however, not how these properties were conceptualised. They are used in an explicative and descriptive fashion to promote understanding of sensemaking processes, rather than a prescriptive or normative specification.

Grounded in Identity Construction

The first and most important property is that sensemaking has as a basis the process of identity construction. Weick et al. (2005, p. 416) quote Mills (1959, p. 55) in referring to identity construction, which “is at the root of sensemaking and influences how other aspects or properties of the sensemaking process are understood”.

He reiterates, “From the perspective of sensemaking, who we think we are (identity) as organisational actors shapes what we enact and how we interpret, which affect what outsiders think we are (image) and how they treat us, which stabilizes or destabilises our identity.”

Fundamental to the process of sensemaking is the assumption that there is an actor (in this case an individual) engaging in making the sense. Since sensemaking is an *act* this also presupposes agency on the part of the sensemaker. This focus on the actor does not, however, refer to a “unified sensemaker”. The concept “sensemaker” does not therefore refer to a singular actor, but rather to a “parliament of selves”¹².

These multiple self-conceptions and the dynamic interaction between the sensemaker

¹¹The work in this section will be based on Weick's seminal work on sensemaking (Weick, 1995) unless otherwise specified.

¹²Mead in Weick (1995, p. 18).

and the environment (as well as him/herself as text) results in identity construction being a discursive, interactive process.

The self (or identity) is the centre from which sensemaking is done. This gives identity definition a primary role in the sensemaking process, since it has a profound impact on the sensemaker's experience of reality itself. Any changes to the sensemaker's self-conception has a resulting impact on the experience of any external reality.

This also transcends the individual and includes his/her context. In the case of persons working for organisations, not only other people's perceptions about them, but also the perceptions about their organisation come into play.

Dutton & Dukerich (1991, p. 548) put it best,

“Individuals' self-concepts and personal identities are formed and modified in part by how they believe others view the organisation for which they work. . . . The close link between an individual's character and an organisation's image implies that individuals are personally motivated to preserve a positive organisational image and repair a negative one through association and disassociation with actions on issues.”

Erez & Earley (1993, p. 28) list three needs people have when managing their own self-conception: “(1) the need for *self-enhancement*, as reflected in seeking and maintaining a positive cognitive and affective state about the self; (2) the *self-efficacy* motive, which is the desire to perceive oneself as competent and efficacious; and (3) the need for *self-consistency*, which is the desire to sense and experience coherence and continuity.”

The influence of this external evaluation of the sensemaker by others (or at least the sensemaker's perception thereof) is paramount to the process of identity construction. The image used in this regard is that of the self in the mirror (or looking glass). This image is taken from a piece by Cooley (1902, p152-153) and is reproduced by Weick (1995, p. 21) at length:

As we see our face, figure, and dress in the [looking] glass, and are interested in them because they are ours, and pleased or otherwise with them according as they do or do not answer to what we should like them to be; so in imagination we perceive in another's mind some thought of our appearance, manners, aims, deeds, character, friends, and so on, and are variously affected by it.

A self-idea of this sort seems to have three principal elements: the imagination of our appearance to the other person; the imagination of his judgment of that appearance; and some sort of self-feeling, such as pride or mortification. The comparison with a looking-glass hardly suggests the second element, imagined judgment, which is quite essential. The thing that moves us to pride or shame is not the mere mechanical reflection of ourselves, but an imputed sentiment, the imagined effect of this reflection upon another's mind. This is evident from the fact that the character and weight of that other, in whose mind we see ourselves, makes all the difference with our feeling. We are ashamed to seem evasive in the presence of a straightforward man, cowardly in the presence of a brave one, gross in the eyes of a refined one, and so on. We always imagine, and in imagining share, the judgments of the other mind.

Sensemaking in the individual is triggered by a failure to confirm the self. This is in direct conflict with the basic function of identity construction aiming to maintain a positive, consistent self-conception. Given the break down of this function, the individual is forced into a sensemaking mode.

Identities are further investigated by projecting them into the environment and observing the environmental responses to these identities. This once again points to the sensemaker not operating in a vacuum, but rather as situated within a specific context. This results in sensemakers simultaneously shaping their environments and reacting to them, thus leading to an interplay between the sensemaker and the environment.

Since this process of identity construction is inherently self-referential, the self (itself) becomes the text being constructed and interpreted in the sensemaking process. The added implication hereof is that the more selves the sensemaker has access to, the more resilient this actor becomes in the face of unexpected environmental cues¹³.

Retrospective

The concept of retrospective sensemaking is an artefact of the heritage of ethnomethodology and specifically the work done by Schutz (1967) on the notion of a “meaningful lived expe-

¹³This of course also speaks to the systems theory concept of *requisite variety* as discussed by Ashby (1956).

rience”. The key is the past-tense use of the word *lived*, immediately making sensemaking reliant on historical observation.

Weick (1995) argues that all human experience (and consequently sensemaking cues) can only exist in the past with humans only becoming aware of *anything* after the sense-act has been completed (even if this is only an infinitesimally small amount of time after the cause of that sense-experience)¹⁴.

In the words of Pirsig (p. 82)¹⁵ “Any intellectually conceived object is always in the past and therefore unreal. Reality is always the moment of vision before intellectualisation takes place. There is no other reality”.

The implication of this is that all sensemaking can only refer to objects present in memory and accessing that memory is a conscious process initiated by the sensemaker. The sensemaker has to cast his/her attention back and select certain memories to act as cues to be placed within his/her sensemaking framework. This addresses the issues of hindsight bias by arguing that it is not a bias, but the only way through which access can be gained to experience of the real world.

The difficulty is that past cues are only selected based on the need for them, given the current sensemaking (or rationalisation) process. This in effect means that sensemaking is not only retrospective, but also retroactive. Since any past experience (or more accurately, “recollection of a past experience”) is influenced by the sensemaker’s current identity, the *Weltanschauung* of the sensemaker becomes the lens through which the past (and thus any) experience is perceived.

Since the make-up of the sensemaker’s frames is constantly evolving and changing, several differing interpretations of the same event may result, forcing the sensemaker to engage in a synthesis of these different sensemaking products. Too many meanings, rather

¹⁴At this point it can be argued that this phenomenology (and the resultant ontological implications) are not unproblematic. In effect Weick is arguing against the existence of the present, by separating sense and perception and introducing a compulsory (conceptual) lag. This conceptualisation also assumes a directly linear process within the stimulus-response (S-R) framework. Additionally he assumes a defined, discrete temporal aspect (quantum) to these sense perceptions. This discussion once again is not within the scope of this thesis, but should receive attention in a critique of the theory. Extensive work in this field was done by many philosophers with Baudrillard as a possible starting point.

Gioia & Mehra (1996) further criticises the notion of sensemaking only being possible in one direction and argue that *prospective* sensemaking must also be possible and is in frequent use. “. . . if sensemaking were not retrospective, we would be forever incapable of making sense of our past - whether real or imagined. And if sensemaking were not also prospective, we would be forever at a loss when asked where we want to go”.

¹⁵as cited in Winokur (1990) cited in (Weick, 1995, p. 24)

than too few, becomes the challenge here.

The danger for ‘objective’ sensemaking is that the sensemaker quite often selects historical cues to match the rationalisation he/she have already constructed for the problematic situation. This act inverts the linear cause-effect (or in this case stimulus-response) mechanism.

Weick (1995) counters this by arguing that in many cases there will be a short (or very short) time separation between the sensory experience logged in memory and the interpretation of that cue, thus reducing the contamination effect¹⁶.

In short the sensemakers’ awareness of the cue is influenced by their attention focus (what they look at), how far they look back and how well they remember or choose to remember.

The retrospective nature of sensemaking is perhaps nowhere better illustrated than in Weick’s famous formulation “how can I know what I think until I see what I say?”.

Enactive of Sensible Environments

In this context, the term *enactment* is used to refer to the fact that “people often produce part of the environment they face” (Weick, 1995, p. 30).

By assuming that sensemaking is an enactive activity the distinction between the observer and an external environment is weakened. This open-system view acknowledges that there is a continuous exchange between the observer and his/her environment. Through acting (living “in the world”) sensemakers create the opportunities and constraints they face.

This *living* or *organic* view of systems is also found in the concept of *autopoiesis* as expressed by Maturana & Varela (1980). Here the focus is on the interplay between structure and function in the self-creating system. In fact, they would go as far as to argue that there cannot be anything outside the system, thereby severely problematising boundary judgement questions and the system/environment distinction¹⁷.

¹⁶This argument, however, displays an inherently platonic character inconsistent with the constructivist approach taken in the rest of the sensemaking conceptualisation. Externally judging “contamination” presupposes some true or correct interpretation.

¹⁷Maturana & Varela did not condone the use of their biological concept in social contexts, or at least did not design the concept to be used in that fashion. However, subsequent to their formulation of their theory many authors, for example Luhmann, have applied this concept in the social sciences.

Another implication of this view is that simply observing the ‘environment’ constitutes an act in and of itself, thereby possibly altering the very context under observation¹⁸.

Follett (1924, p. 118-119) casts this as an auspicious situation emancipating the observer, whilst at the same time constraining him/her: “we are neither the master nor the slave of our environment. We cannot command and the environment obey, but also we cannot, if we would speak with the greatest accuracy, say that the organism adjusts itself to the environment, because it is only part of a larger truth.”

Once again this does not adhere to the traditional stimulus-response framework. At the very least the notion of one-way linear causality must be abandoned. This also has implications for the Heideggerian conception of ‘thrownness’ earlier applied by Weick. Given the enactive nature of the sensemaking process, it raises the question of what exactly is meant by the ‘situation’ or ‘world’ into which the sensemaker is ‘thrown’. Perhaps one could even extend the metaphor to ask whether the sensemaker is now throwing him/herself?

Enactment is also accompanied by the concept of ‘bracketing’. This refers to the fact that sensemakers find themselves in a stream (or *flow*) of sense perceptions when observing the world. This stream is then actively interrupted (punctuated) with specific observations being selected and conceptualised upon (‘bracketed’). Once again the notion of Platonic Forms is rejected with the observer being the final arbiter in the definition of concepts under consideration.

Of course this also means that the enactive process can be both constructive and destructive, depending on whether the observer chooses to focus on a specific sensory input or not.

Social

As a rule, human conduct is shaped by others. Due to the inherently social nature of the sensemaker (and the understanding that his/her conception of reality is influenced by his/her conduct and actions) we can conclude that sensemaking is almost always contingent on other actors. This makes the notion of individual sensemaking a bit of a misnomer.

March & Heath (1994, p. 210) argues that meaning is dependent on social interaction and takes both its coherence and its contradictions from this social basis.

¹⁸cf. Information Systems version of the ‘Observer Effect’.

Even when persons are functioning on their own in an individual capacity, their actions often cast others (or perceived others) as the audience, thereby involving them in the sense-making act. This, however, does not mean that all action is social or contributes to shared meaning - the implication here is restricted to the social nature of sensemaking. Society (or community) is constructed by more than 'shared meaning', and 'shared experiences' could prove much more important for social cohesion than collective sensemaking. Therefore, even though sensemaking is a decidedly social act, it is not necessarily a collective activity. Given the social nature of sensemaking, the issue of communication also becomes more relevant.

Since the agents of sensemaking (*sensemakers*) are social beings, the acts which they perform (*sensemaking*) cannot be presumed to be situated in a vacuum devoid of social interaction. Sensemaking is not solitary because the internal functions of the sensemaker are contingent on others.

A social nature should not be construed as a necessarily cooperative one. The concept 'society' should be read with all the component power relations, coercive acts and politics implied by it. Sensemaking therefore quite often means a pragmatic *modus vivendi* applied by several actors, rather than one harmonious corporation.

Sensible meaning is meaning for which there is social support, consensual validation and shared relevance. The judgement on the sensibility of meaning is made by the individual sensemaker with due consideration for the social structure and community with which that sensemaker associates him-/herself.

To change meaning becomes equated with changing social context. This could take the form of changing the sense prompt being evaluated, the conceptualisation of the community, the social context itself (through an interactive process) or the interpretation of the perceived responses to the sensemaking product.

Ongoing

The sensemaker experiences life as a constant, continuous flow of sensations. Since this flow does not stop and contains an endless stream of sensory data, coping and processing these data is largely an autonomous process. Certain unexpected observations can, however, focus the individual's attention and interrupt this flow.

As soon as these interruptions are noticed the conscious act of sensemaking is initiated.

Due to the dynamic nature of the stream of experience, sensemaking itself becomes a continuous process without a defined start or end. Since sensemaking also takes place at the fundamental level between “perception” and “sense” this process becomes part of the most basic awareness of the individual.

Sensemaking is prompted by interruptions in the flow of experience. This interruption causes “arousal” (as in a nervous system) and elevates the specific issue to the level of consciousness, requiring active sensemaking to be resolved.

Weick (1995, p. 43) refers us to Dilthey in saying “there are no absolute starting points, no self-evident, self-contained certainties on which we can build, because we always find ourselves in the middle of complex situations which we try to disentangle by making, then revising, provisional assumptions.” This brings sensemaking into the realm of the hermeneutic.

Continuing in the same philosophic vein, Weick (1995, p. 44) references a summary of Heidegger’s concept of thrownness by Winograd & Flores (1986, p. 34-36) (as briefly dealt with in the previous section and now quoted in full):

1. You cannot avoid acting: Your actions affect the situation and yourself, often against your will.
2. You cannot step back and reflect on your actions: You are thrown on your intuitions and have to deal with whatever comes up as it comes up.
3. The effects of action cannot be predicted: The dynamic nature of social conduct precludes accurate prediction.
4. You do not have stable representation of the situation: Patterns may be evident after the fact, but at the time the flow unfolds there is nothing but arbitrary fragments capable of being organised into a host of different patterns or possibly no pattern whatsoever.
5. Every representation is an interpretation: There is no way to settle that any interpretation is right or wrong, which means an “object analysis” of that into which one was thrown, is impossible.
6. Language is action: Whenever people say something, they create rather than describe a situation, which means it is impossible to stay detached

from whatever emerges unless you say nothing, which is such a strange way to react that the situation is deflected anyway.

Although it is clear that the flow of experiences or sensory data can be interrupted, the notice paid and meaning assigned to these interruptions (as well as the labelling thereof) is dependent entirely on the specific sensemaker involved. The selection of triggers for sensemaking is not uniform or universal.

Since individuals are also involved in several “projects” at any given time, the attention they pay and meaning they assign will also be influenced by the focus required to successfully function within these projects.

Focused on and by Extracted Cues

Earlier sections dealt with the process of sensemaking starting with cues being placed within a mental frame of reference. This section will deal with this process in a bit more detail.

From the stream of experience, people select certain data (datum-points) for further intellectualisation. James (1895) (as referenced in Weick (1995)) points out that this process of extracting cues is the cornerstone of the sensemaking process. In fact, people give so much weight to these extracted cues that they equate the cue with the context from which the cue was obtained. The extracted nature of the cue also focuses the mind to consider the implications more carefully than when dealing with the entire context.

The mechanism is thus to use these smaller, more manageable cues to reach conclusions about the whole under consideration. In doing so, these cues become reference points with which the sensemaker approaches the reality he/she faces. Since this reference point can be of great importance, the selection process is one of control.

These cues are extracted from the stream of experiences and are therefore selected by a specific observer for further attention. Weick points out that the same cue¹⁹ can have multiple interpretations depending on the observer dealing with that cue.

The context of the cue and the mental models of the observer determine the interpretation and understanding of the cue. This makes all cues highly contextual and involves the cue’s context as part of the sensemaking “frame” applied by the sensemaker.

¹⁹Weick does not, however, convincingly argue how the cue can exist as an objective object outside the observer.

We have previously discussed the mechanism prompting the sensemaking act. Cue extraction is intertwined with this process and labeled *noticing*. Starbuck & Milliken (1988, p. 265-266) refer to noticing as “activities of filtering, classifying, and comparing” and contrast this with sensemaking as “interpretation and the activity of determining what the noticed cues mean.” They add “[s]ensemaking focuses on subtleties and interdependencies, whereas noticing picks up major events and gross trends. Noticing determines whether people even consider responding to environmental events. If events are noticed, people make sense of them; and if events are not noticed, they are not available for sensemaking”.

Since these cues are all situated in a (social) context, politics and power play an important role in the determining and extraction of cues. When talking about noticing and sensemaking one needs to constantly bear in mind that there are real human-beings driving these processes. The context will determine which cues become salient and which are ignored (or just not noticed).

An extracted cue (as part of the sensemaking process) have profound implications on behaviour. Specifically its function as a *reference point* is of great importance. Interestingly the exact content of the cue or even its interpretation is secondary to the mere *existence* of the reference point. This again confirms the primacy of action in the sensemaking context. “But regardless of the cues that become salient as a consequence of context, and regardless of the way those extracted cues are embellished, the point to be retained is that faith in these cues and their sustained use as a reference point are important for sensemaking” Weick (1995, p. 53). These reference points help to create order and give people a starting point from which to proceed.

“Once people begin to act (enactment) they generate tangible outcomes (cues) in some context (social), and this helps them discover (retrospect) what is occurring (ongoing), what needs to be explained (plausibility), and what should be done next (identity enhancement).” (Weick (1995, p. 55) quoting Starbuck (1993)).

Driven by Plausibility Rather Than Accuracy

Sensemaking de-prioritises the normal problem solving goal of accuracy and substitutes it for plausibility. The goal in this process is not to discover the “perfect underlying truth” (since this is accepted not to exist), but simply to find some way of coping with the reality presented. Weick (1995, p. 56) states, “accuracy is nice, but not necessary” and “the

sensible need not be sensible”.

The rejection of the objective perception means sensemaking is about “plausibility, pragmatics, coherence, reasonableness, creation, invention, and instrumentality” (Weick, 1995, p. 57). The “accuracy” of the recollections are, therefore, instrumental and not related to some externalised reference point.

The approach conforms to the systems thinking principle of *satisficing*. Isenberg (1986, p. 242-243) expands,

“Plausible reasoning involves going beyond the directly observable or at least consensual information to form ideas or understandings that provide enough certainty. . . There are several ways in which this process departs from a logical-deductive process. First, the reasoning is not necessarily correct, but it fits the facts, albeit imperfectly at times. Second, the reasoning is based on incomplete information.”

In short, total accuracy is not only impossible given the rejection of the platonic world-view, but also simply not needed. Even in the few cases where accuracy could be improved it very seldom justifies the additional expense in terms of time and resources. Fiske (1992) describes this unavoidable trade-off and notes that managers almost always favour speed.

When focusing on plausibility it is also quite often found that what is plausible for one group may be totally implausible for another. Mills (2003, p. 169-173) lists a few factors that are important to ensure the plausibility of an interpretation including:

1. tapping into an ongoing sense of the current climate
2. consistency with other data
3. facilitating ongoing projects
4. reducing equivocality
5. providing an aura of accuracy
6. offering a potentially exciting future

Often, the inaccuracy (combined with the awareness the actors have of this limitation) can be beneficial since this opens the possibility for highly dynamic and creative solutions

without the constraints of measuring all possible solutions against a defined “reality”. Limited accuracy also focuses the actors’ minds to be aware of feedback loops and monitor interventions since the results (even in relatively simple systems) will be unpredictable.

Since sensemakers operate in complex and complicated environments, a perfectly accurate representation would contain high levels of detail and an enormous amount of data. These data can be very overwhelming and paralyse decision makers slowing down the process even more. The abundance of possible sense data get transformed into *noise*. This case is also displayed in the cognitive dissonance experienced by actors operating in situations of ambiguity.

A central part of the sensemaking process is the embellishment and expansion of extracted cues. These expansions lead to multiple interpretations and occasions for sensemaking. Mediating these interpretation to get to an “accepted accurate view” would prove impossible in all but the most trivial matters. The interpretation process starts at the moment of arousal and is a dynamic process that develops as the context gets augmented with new experiences. Delaying action until a stable authoritative view has been determined would mean a permanent delay; unless the system is in a stable zone with no new inputs being received.

We have also established that the sensemaking process (and in fact all sense perception) is a retrospective process operating from the memory of experienced events. Since memory is a function of an actor (and thus constructed within that context), the memories themselves cannot be entirely stable or “accurate”.

Weick et al. (2005, p. 415) summarise:

Sensemaking is not about truth and getting it right. Instead, it is about continued redrafting of an emerging story so that it becomes more comprehensive, incorporates more of the observed data, and is more resilient in the face of criticism.

4.2.4 Characteristics of Sensemaking

We have now dealt with the classic *Seven Properties* of sensemaking. As noted earlier, these properties relate to the sensemaking process, but are not necessarily enough to address all of the main themes that come into play when studying *organisational* sensemaking. Sensemaking and organisation are intrinsically linked: “Organisation is an attempt to order

the intrinsic flux of human action, to channel it toward certain ends, to give it a particular shape, though generalising and institutionalising particular meanings and rules.” (Tsoukas & Chia, 2002, p. 570). Given this view of organisation, the correlation with sensemaking is obvious.

To this end the properties are expanded by Weick et al. (2005) to include a few “characteristics”. This section outlines a few salient points from this article. In many places these notes are simply expansions of the thoughts noted in the “properties” description and in these cases they will of course not be repeated.

In short they identify the following characteristics:

1. Sensemaking Organises Flux
2. Sensemaking Starts with Noticing and Bracketing
3. Sensemaking is about Labelling
4. Sensemaking is Retrospective
5. Sensemaking is about Presumption
6. Sensemaking is Social and Systematic
7. Sensemaking is about Action
8. Sensemaking is about Organising through Communication

Sensemaking Organises Flux

The starting point in the sensemaking process is that the sensemaker is confronted with chaos. This chaos is confusing and activates autonomic arousal in the sensemaker, triggering the sensemaking process. We start with “an undifferentiated flux of fleeting sense-impressions and it is out of this brute aboriginal flux of lived experience that attention carves out and conception names” (Chia, 2000, p. 517).

Sensemaking is thus a systemic process of organising — confronting the chaos and noticing, selecting and bracketing observations to try and frame those phenomena within a framework that enables the observer to (i) cope and (ii) make sense of their experience. Weick et al. (2005, p. 410) identify the central theme of sensemaking to be “people organise

to make sense of equivocal inputs and enact this sense back into the world to make the world *more orderly*” (emphasis added).

Sensemaking Starts with Noticing and Bracketing

If sensemaking originates from an unordered chaotic flux, the mechanism for identifying salient observations from that stream is non-trivial. Paying attention to all jarring, unexpected and arousing stimuli is impossible for several reason (not the least of which is the enormous amount of triggers observed by the sensemaker). Chia (2000, p. 517) notes that triggers “have to be forcibly carved out of the undifferentiated flux of raw experience and conceptually fixed and labeled so that they can become the common currency for communicational exchanges”.

The act of interpretation can only start once the object of interpretation has been identified (and delimited). The noticing and bracketing process draws the boundaries around stimuli, enabling their conceptualisation in the next step of sensemaking. These bracketing acts are of course not objective or universal. The mental models, framework and *Weltanschauung* of the sensemaker will have a profound effect on the boundary judgements effected. In essence the bracketing act already starts to reduce complexity and select salient points to enable the further ordering of the world.

Sensemaking is about Labelling

Once specific stimuli have been selected from the stream of experience, the sensemaker proceeds to label and categorise these stimuli. This labelling process is, however, not only a personal ‘sorting-out’. Labelling stimuli attach more generalised attributes to them and allows other actors to recognise and react to these stimuli.

Labelling is a further simplification process and often calls on set categories (in the Kantian sense) to order and organise. The assigned labels transcend the individual sensemaker and by doing so further the objectification of the perceived phenomena, enabling other actors to relate to them and interact with them. “For an activity to be said to be organised, it implies that types of behaviours in types of situation are systematically connected to types of actors... An organised activity provides actors with a given set of cognitive categories and a typology of action.” (Tsoukas & Chia, 2002, p. 573). Note that the authors again emphasise the systematic nature of the activity.

Although these categories or labels are not simply attributes of the specific phenomena under observation and have general application, it should be noted that these labels are still in essence social constructs pliable to the sensemaker's needs. This property is referred to as *plasticity* (Weick et al., 2005, p. 411).

Two phenomena sharing the same label also does not imply that these phenomena are identical. These labels are simply tools sensemakers use to get a grip on the myriad of phenomena continuously being observed.

Sensemaking is about Presumption

Weick refers to the mechanism used by the sensemaker to start the interpretation phase as *presumption*. In essence the sensemaker observes a phenomenon, selects it, brackets it, labels it and then proceeds to state some possible hypothesis as to the relevance and effect of that observation. This is the presumption.

Although the sensemaking process is indeed systematic, it is not entirely rational or premeditated. In line with a satisficing approach the interpretation of cues is a pragmatic, immediate process that is adaptive to additional inputs and analysis.

As more information is gathered and interpretation progresses this initial presumption is modified to incorporate the ongoing sensemaking process. This presumption is continually adjusted through progressive approximations. Paget & Cassell (2004, p. 143) explain²⁰ “The...work process unfolds as a series of approximations and attempts to discover an appropriate response. And because it unfolds this way, as an error-ridden activity, it requires continuous attention...”

Sensemaking is about Action

Several of the preceding sections have dealt with sensemaking being an inherently action-driven activity. To understand the concept of sensemaking is to understand the intricate interplay between *action* and *talk*.

Talk is imperative due to the social and collective nature of sensemaking in the organisational context. The continuous adaptation and adjustment process is heavily influenced by talking about the actors' current understanding and by listening to the comments on

²⁰In the context of medical work as referenced by Weick et al. (2005).

the activities undertaken.

Weick et al. (2005, p. 412) elegantly expand:

“In sensemaking, action and talk are treated as cycles rather than as a linear sequence. Talk occurs both early and late, as does action, and either one can be designated as the ‘starting point to the destination’. Because acting is an indistinguishable part of the swarm of flux until talk brackets it and gives it some meaning, action is not inherently any more significant than talk, but it factors centrally into any understanding of sensemaking.”

The retrospective nature of sensemaking also means that the past is extremely important to contextualise the present, while simultaneously being constantly adapted. The past is thus *at the same time* being both “honoured and rejected”.

Sensemaking is about Organising through Communication

Talking has already been noted as an important theme as it relates to action, but talking (and communication more generally) is central to sensemaking itself. In the organisational context sensemaking will often occur collectively. This presupposes communication between the different individuals involved in the sensemaking effort.

“The image of sensemaking as activity that talks events and organisations into existence suggests that patterns of organising are located in the actions and conversations that occur on behalf of the presumed organisation and in the texts of those activities that are preserved in social structures.” (Weick et al., 2005, p. 413).

Weick proceeds to introduce the concept of *articulation*, defined as “the social process by which tacit knowledge is made more explicit or usable”. The communicative function here thus refers to more than a simple sharing of self-evident facts or murmurations about some agreed upon interpretation of the current cues. Through communication and talking the actors in the collective sensemaking process elevate the underlying, tacit and obscured meaning of their observations and bring it to discussion by making it public, labelled and explicit (Obstfeld, 2004). In doing so they make these bracketed observations accessible to the group and enable engagement with the underlying issues.

This discussion or ‘talking about’ should also not be seen as a mere transfer of information. The discussion process itself has a dialectical element with the talker also engaging

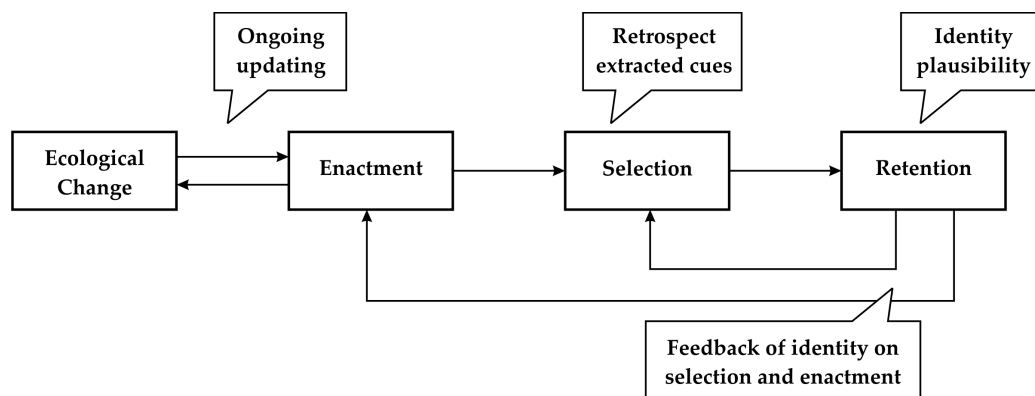


Figure 4.1: *The Relationship Among Enactment, Organising, and Sensemaking*

SOURCE: Jennings & Greenwood (2003)

reflexively and adapting his/her own understanding of the situation both through telling and by reacting to the response of the actor hearing him/her talk.

By explaining a thought process not only the audience becomes aware of the sense-maker’s thoughts, but the underlying thoughts also become clear to the person doing the talking.

4.3 Sensemaking as Enactment

Systems (in this case organisations) are constantly interacting with and responding to their environment. Weick et al. (2005) acknowledge this interaction and proceed to apply the *enactment theory framework* of Campbell (1987, 1997) to sensemaking.

Weick et al. (2005, p. 414) summarise work done by Jennings & Greenwood (2003) explaining “enactment theory” (this is also visually represented in their model (p. 132) reproduced in Figure 4.1.):

Sensemaking can be treated as reciprocal exchanges between actors (Enactment) and their environments (Ecological Change) that are made meaningful (Selection) and preserved (Retention). However, these exchanges will continue only if the preserved content is both believed (positive causal linkage) and doubted (Negative causal linkage) in future enacting and selection. Only with ambivalent use of previous knowledge are systems able both to benefit from lessons learned and to update either their actions or meanings in ways that adapt to

changes in the system and its context.

In this view, organisation is presented as a sequence: change — enactment — selection — retention.

Ecological change and *enactment* are intertwined. These steps relate to arousal, selecting phenomena from the stream, starting to order the flux, investigating external influence and noticing/bracketing. These activities lead into the *selection* phase, focusing on the simplification of perceived sensory inputs, articulation of the underlying issues and reducing the narrative to make it more manageable. Finally the sensemaking process moves towards *retention* in an attempt to maintain the plausible.

This process sequence illustrates that making sense of equivocality is to “enact order back into the world” (Weick, 1969, p. 40-42).

4.4 Occasions for Sensemaking

Sensemaking is brought into focus whenever the perceived world differs from the expected one. As soon as the observer is confronted with an unexpected realisation the continuity of the flow of experiences is breached. This breach leads to cognitive dissonance and the feeling that reality is disorganised, prompting the sensemaker to engage in actions aimed at resolving the discrepancy between the expected and the perceived.

To understand sensemaking, it is important to realise that this act is not always a conscious project, but rather a normal part of experiencing the world. Gioia & Mehra (1996, p. 1228-1229) point out that the largest part of organisational life carries on without the actor actively engaging or giving it his/her full attention. The actor’s sense can be “modified in intricate ways out of awareness via assimilation of subtle cues over time”.

Rather than the sensemaker focusing on every single observation, sensemaking is activated when shocks are experienced. A whole variety of these shocks are possible. The recognition of a shock depends largely on our *perceived environmental uncertainty* (Duncan, 1972). This term clearly points to perception as a product of both the environment and the predispositions of the observer.

Huber & Daft (1987) identified three properties instrumental to sensemakers taking notice of stimuli:

- *Information load* — people are confronted with problems related to the quality, ambiguity and variety of information, and have to take steps to deal with this problem.
- *Complexity* — it is exceedingly difficult to deal with information in context where the relations and interactions between elements are unknown and unknowable.
- *Turbulence* — turbulence is a function of instability (frequency of change) and randomness (frequency and direction of change).

Weick proceeds to identify two types of occasions for sensemaking in organisations - *ambiguity* and *uncertainty*.

4.4.1 Ambiguity

Ambiguity²¹ is the shock of confusion. In ambiguous situations actors are confronted with confusing stimuli open to multiple interpretations and several plausible explanations. Although the reflex response in these cases is often to gather more information, this is not helpful in cases of ambiguity. The abundance of conflicting information does not meet the criteria for rational decision making and thus invalidates the default response mode.

March & Heath (1994, p. 178) provide a good description:

“Ambiguity refers to a lack of clarity or consistency in reality, causality, or intentionality. Ambiguous situations are situations that cannot be coded precisely into mutually exhaustive and exclusive categories. Ambiguous purposes are intentions that cannot be specified clearly. Ambiguous identities are identities whose rules or occasions for application are imprecise or contradictory. Ambiguous outcomes are outcomes whose characteristics or implications are fuzzy.”

To illuminate the characteristics of ambiguous situations a summary of these scenarios can be found in Table 4.1.

²¹Weick (1995, p. 94) points out that the term *ambiguous* can be quite confusing since in common use it also refers to situations lacking clarity. To avoid this he advocates use of the term *equivocal*. Regardless, the bulk of his text still uses the label ‘ambiguity’.

Table 4.1: *Characteristics of Ambiguous, Changing Situations*

Characteristic	Description
Nature of problem is itself in question	“What the problem is” is unclear and shifting. Managers have only vague or competing definitions of the problem. Often, any one “problem” is intertwined with other messy problems.
Information (amount and reliability) is problematic	Because the definition of the problem is in doubt, collecting and categorising information becomes a problem. The information flow threatens either to become overwhelming or to be seriously insufficient. Data may be incomplete and of dubious reliability.
Multiple, conflicting interpretations	For those data that do exist, players develop multiple, and sometimes conflicting, interpretations. The facts and their significance can be read in several different ways.
Different value orientations, political/emotional clashes	Without objective criteria, players rely more on personal and/or professional values to make sense of the situation. The clash of different values often charges the situation politically and emotionally.
Goals are unclear, or multiple and conflicting	Managers do not enjoy the guidance of clearly defined, coherent goals. Either the goals are vague, or they are clearly defined but contradictory.
Time, money or attention are lacking	A difficult situation is made chaotic by severe shortages of one or more of these items.
Contradictions and paradoxes appear	The situation has seemingly inconsistent features, relationships, or demands.
Roles are vague, responsibilities are unclear	Players do not have a clearly defined set of activities they are expected to perform. On important issues, the locus of decision making and other responsibilities is vague or in dispute.
Success measures are lacking	People are unsure what success in resolving the situation would mean, and/or they have no way of assessing the degree to which they have been successful.
Poor understanding of cause-effect relationships	Players do not understand what causes what in the situation. Even if sure of the effects they desire, they are uncertain how to obtain them.
Symbols and metaphors used	In place of precise definitions or logical arguments, players use symbols or metaphors to express their points of view.
Participation in decision-making is fluid	Who the key decision makers and influence holders are changes as players enter and leave the decision arena.

4.4.2 Uncertainty

Uncertainty is the shock of ignorance. In contrast with ambiguity, uncertain situations are characterised by a lack of information needed to make an informed decision. This lack of data makes it difficult to reach satisfying interpretations, leaving the sensemaker in a state of shock.

Three types of uncertainty were identified by Milliken (1987):

- *State uncertainty* — The sensemaker is unsure of how components of the environment

are changing.

- *Effect uncertainty* — The impact of environmental changes, on the system, is unclear.
- *Response uncertainty* — The options open to the system is not known.

4.5 Level of Analysis (Institutional Theory)

Sensemaking occurs at both the organisational level and at the level of the individuals present in the organisation. When analysing sensemaking practices one should note that these two levels are obviously related, but they are not identical.

We have already seen that sensemaking and organising (as an activity) are related - thus there are implications for organisations as structures. Weick (1995, p. 36) comments, “sensemaking is the feedstock for institutionalisation”.

When looking at sensemaking in organisations, the macro and micro levels of analysis should be brought into relation. Hedstrom & Swedberg (1998, p. 21) argues that we can explain change at the macro level if we show “how macro states at one point in time influence the behaviour of individual actors, and how these actions generate new macro states at a later time”. The argument about the impact of sensemaking on institutional structure will be expanded upon in a later chapter.

Weber (2003) looked at globalisation and connected sensemaking and macro institutional perspectives. To this end he argued that corporate agendas and attention is often directed by public actors - this introduces another perspective (and actor) into the sense-making process.

The concept “organisation” is unfortunately a misnomer. Most organisations are not monolithic structures, but rather segmented collaborations between many actors (Weick, 2001). These organisations also do not operate in a homogeneous environment.

4.5.1 Levels of Sensemaking

Excluding the most obvious and localised sensemaking processes at the individual (*intra-subjective*) level, three higher levels of sensemaking are identified (Wiley, 1988). These are the *intersubjective*, *generic subjective* and *extrasubjective*.

Intersubjective

The intersubjective level of sensemaking goes beyond the intrasubjective's individual thoughts, feelings and intentions. All these elements are combined in a process moving the “I” to a “we” through conversing. This causes the subjective meaning to be interchanged and synthesised with the subjective meaning of another self.

In essence this results in a fusion of meaning between two or more selves within the context of some social interaction. This creates a “level of social reality” (p. 254).

Generic Subjective

The generic subjective is on the level of social structures. The focus on this level is no longer placed on individual (concrete) human beings. “Selves are left behind at the interactive level. Social structure implies a generic self, interchangeable part — as filler of roles and follower of rules — but not concrete, individualised selves. The ‘relation to subject’ then, at this level is categorical and abstract” (Wiley, 1988, p. 258).

Many social phenomena such as work roles, relational roles, social networks, “scripts” and “plots” are located at this level. These standardised plots make people interchangeable and enable organisations to function given a constantly changing workforce.

In managing uncertainty, sensemakers rely on a mixture of the intersubjective and generic subjective with the intersubjective acting as a fall-back mechanism to fill gaps in sense.

Extrasubjective

The highest level of sensemaking is that of the extrasubjective. This is the level at which culture is located. On this level the generic self is replaced by “pure meaning” (Popper, 1968).

This level of *culture* is an “abstract idealised framework constructed as the result of many cycles of prior interaction” (Weick, 1995, p. 72).

When proceeding with a sensemaking analysis, it is imperative to consider the appropriate level of sensemaking for that particular analysis. Also note that sensemaking processes may occur simultaneously on different levels.

This concludes the brief summary of sensemaking, but is in no way meant to be complete. Additional concepts applicable to specific analyses will be introduced in the following chapters. At this point enough of a (sensemaking) framework has been established to enable the evaluation of agile development in this light.

Chapter 5

Agile Development as Managed Sensemaking

Previous chapters have evaluated the development of software methodologies, defined the term ‘agile development’ and presented an integrated view of sensemaking theory. As stated in the first chapter, the goal is now to evaluate our conception of agile development against the background of this integrated view of sensemaking and thereby gain a better understanding of this new approach to software development¹.

The objective in this case is to try and establish whether the underlying principles and assumptions in the agile approach can be explained by the process of sensemaking (or rather, *organisational sensemaking*). Through evaluating stated values and principles and investigating the activities of agile development, this chapter proposes that agile development can be better understood from a sensemaking perspective and even more importantly, that the reasons for agile development as approach rather than formalised approaches resonate particularly well with the framework of understanding cognitive processes in changing and unpredictable environments.

Using sensemaking as a conceptual framework is not a novel approach. However, as demonstrated in the previous chapter, this theory was not designed as a formal model against which cases can be evaluated in a check-list fashion. Sensemaking was also not conceptualised with a set of criteria which phenomena need to display in order to qualify as being a process of ‘sensemaking’.

¹Although the salient points are briefly listed here, it is recommended that the reader refer back to chapters three and four for the detailed discussion of agile development and a review of sensemaking.

To perform this analysis we therefore have to look at the entire agile development approach and ask if the behaviour, specifications and underlying assumptions display the same patterns as seen in the literature describing sensemaking acts. To this end a thematic approach is followed, identifying several correlations between sensemaking and agile development.

Sensemaking theory is presented as a scaffolding with which to gain a better understanding of the emergent behaviour observed in the establishment of agile development.

Once again, bear in mind that this thesis is dealing with *managed sensemaking*. By evaluating the fit between sensemaking and agile development, we can reflect on the steps organisations need to take to ensure that the organisational conditions are conducive to the act of sensemaking being undertaken in their own contexts.

At the same time the notion of managing sensemaking *itself* is a misnomer. Attempting to influence sensemaking acts in this way is naïve and futile for the same reasons knowledge management cannot manage knowledge *itself*. Chapter 6 expands on this topic.

5.1 Meta-sensemaking & Recursion

This chapter will outline several correlations between different sensemaking acts and activities within the agile development process. Sensemaking happens at many different levels during agile development. This section will briefly look at these layers of sensemaking.

5.1.1 Level of Analysis

The first instance at which sensemaking comes into play is on the meta-level, in the writing and reading of this thesis itself. The study itself maintains internal consistency by approaching the sensemaking analysis of agile development through a rationalised process of sensemaking.

On the converse (and as an aside) the reader of this thesis will also be engaged in an act of interpretation and sensemaking when trying to understand the meaning of the text presented here.

The most basic and obvious level at which we can see similarities between sensemaking and agile development, is when looking at the individuals involved in the software development process. Regardless of the specific development methodology, these individuals are

still, most basically, human. As human beings they engage and interact with the world as everyone else. These interactions are explained and can be better understood by leveraging the sensemaking paradigm.

This conception of workers is in stark contrast with neoclassical or behaviourist conceptions of employees as interchangeable, mechanical parts in the production process. Instead, workers in this context are valued for their own, individual, idiosyncrasies, views and skills.

Although the sensemaking process can happen independently of any agile development, we need to bear in mind that *individual sensemaking* has a major impact on the worldview of actors - whether it is conscious or not. Sensemaking, in this regard, is simply a descriptive mechanism trying to explain the interactions between human actors and their environment.

The second level at which sensemaking becomes prominent is that of agile development *as a methodology*. This thesis has dealt with that at length. Several key characteristics of agile development can be explained by utilising sensemaking as a framework with which to analyse it. Positing that agile development *as a methodology* is also a higher-level *organisational* sensemaking activity helps us to understand some of the mechanics of this methodology.

A last process worth analysing in terms of sensemaking is the development of agile development itself. Looking at the evolution of software methodologies leading up to agile methodologies (discussed at length in chapter 2), it can be argued that that process itself was a result of the actors' sensemaking endeavours. The route followed to the establishment of the agile manifesto show the authors had a commitment to the hermeneutic world view entrenched in sensemaking practice.

5.1.2 Distinctions

This thesis does not purport that agile development is equal to, or the same as, sensemaking. That would be a terrible misunderstanding of the issues under consideration.

This study, and particularly this chapter, identify certain important patterns present in agile development that fit well with a sensemaking explanation. Sensemaking operates on the general hermeneutic level as a basic cognitive process in individuals (and under certain circumstances in organisations). Agile development, on the other hand, is a specific instance of organisational sensemaking and can therefore be better understood by applying that same framework to analyse the phenomenon.

In terms of ordering these topics, sensemaking can be conceptualised as a cognitive interpretive act with agile development as an instance of organisational sensemaking being applied to software development methodologies.

This chapter will attempt to highlight some of the correlations and patterns leading us to this insight.

5.1.3 Occasions for Agile Development

Agile development is being used as a replacement for traditional software development methodologies. As a result, this new approach is used in planning, new product development and even maintenance functions (as explained elsewhere in this chapter). There are, however, certain types of situations for which agile development is particularly well suited and others where it would be unwise, or dangerous, to switch to this type of methodology (Nerur et al., 2005; Turk et al., 2002).

Agile approaches were developed specifically for a world that is constantly changing and inherently unpredictable. Chronologically these are also second generation approaches, superseding traditional methodologies and thus being applied in contexts where more rigid approaches were not delivering satisfactory results. In essence, under certain conditions traditional approaches were not delivering results corresponding to the presumptions and expectations of the users. This dissonance resulted in a development process leading to the establishment of agile methods (refer to chapter 2).

Here agile development and sensemaking differ. Whereas sensemaking is a generic human activity, the selection of agile development methodologies depends greatly on the task at hand. For projects operating within defined parameters with a rigid design, agile development would not be very well suited. On the other hand products being developed in turbulent conditions, surrounded with uncertainty could gain much from this more flexible approach.

Teleology is key to understanding this difference. Sensemaking is primarily applied to the individual on the most fundamental level and can then be extended to organisational, societal and cultural sensemaking. Agile development, on the other hand, does not aim to be a general hermeneutic framework. It is simply a set of methodologies designed for the efficient development of products under certain conditions.

Agile development is therefore a tool suitable to application in certain contexts, but not

as a general framework. Deciding which occasions warrant the use of agile methodologies is itself a sensemaking exercise.

5.2 Worldview (*Weltanschauung*)

The starting point for this sensemaking analysis is to investigate the underlying assumptions constituting the Worldview (or *Weltanschauung*) prevalent in agile development circles².

5.2.1 De-emphasising the plan

Probably the most obvious refrain detected in all writing about agile development is the de-emphasising of a rigid or formal plan that is to be completed before implementation starts. This is such an important part of the agile approach that one of the four central tenets in the *Agile Manifesto* (Beck et al., 2001) reads:

“Responding to change over following a plan”

This fundamental acceptance of change runs deeper than a simple motto in a manifesto. Nandhakumar & Avison (1999) point out that the agile approach regards formal plans as “fiction” and rejects the notion that such a formalised, fixed course of action can be adequate in the complex world in which software development finds itself. It is this rejection of the notion of the world as “stable” that opens the door to a more flexible and responsive worldview. Williams & Cockburn (2003) go to the extreme when they say agile development is “about feedback and change”. This intrinsic belief is seen throughout the entire development approach.

In rejecting a stable system and de-emphasising central pre-planning, supporters of agile development reject the notion of platonic absolutes. Accepting the world as dynamic, complex and changing has a profound impact on practical aspects of the development process. That is also exactly what is displayed in the agile approach.

The entire approach is developed to be as flexible and adaptive as possible. This is obvious in the short development cycles, increased feedback loops to customers and environment, elevation of the status of functional code, and self-organising structure. All these

²Once again, the term ‘circles’ here does not purport that such a defined group indeed exists, but rather refers to the general patterns that can be gleaned by analysing the material presented here.

characteristics intrinsically accept an unstable world and provide means to cope with that reality.

The second principle of agile development (Beck et al., 2001) states this even more explicitly:

“Welcome changing requirements, even late in development...”

In short, followers of agile development cannot accept that a pre-formulated, static plan is sufficient to cope with an ever-changing world.

The rejection of absolutes (in the post-modernist tradition) also takes shape in the refusal of either agile development or sensemaking proponents to define and stipulate the exact confines of the respective movements. Agile development and sensemaking are both still changing, growing and developing.

The previous chapter discussed at length how sensemaking functions in an environment where actors are open to change and reject an external stable design. Both agile development and sensemaking share in this hermeneutic approach.

5.2.2 Functional pragmatism

The second correlation between the worldviews of sensemaking and agile development is the pragmatic focus. In sensemaking this is referred to as favouring “plausibility” over “accuracy” (Weick, 1995). In agile development this is displayed in the second tenet (Beck et al., 2001) stating:

“Working software over comprehensive documentation”

For developers in the agile paradigm, the only true measure of progress is the demonstration of functional code. (The third principle of agile development refers to “working code” (Beck et al., 2001)).

This commitment to functional pragmatism is central to the design of the Scrum process. The entire development cycle is modified to deliver *working* code after each sprint (maximum 30 days) (Schwaber, 1995). This discipline is maintained by *releasing* all code into production after each sprint.

Extreme programming (XP) of course also displays the same short development cycles and emphasis on working code, but also expands this focus by elevating testing as the final

decider on the success of any step (Beck, 2000, 2004). Since many different interpretations of the problem solution exist in any product, the actual code is not (and cannot be) evaluated against some idealised design. Testing becomes the mechanism through which the required measure of objectivity can be achieved.

There are two additional motivations for the preference of plausibility (sufficiency) over accuracy in the agile process. As referred to earlier, Fiske (1992) points out that there is always a trade-off between speed and accuracy. The entire agile model is constructed to reduce development time in order to maximise agility. Time and resource-intensive tasks such as documentation are minimised. In short, if written code passes the automatic testing it has satisfied the minimum requirements agreed upon and can be regarded as sufficient.

The acceptance of the world as changing and dynamic (discussed earlier) also negates the usefulness of extensive documentation and “aesthetically pleasing” code. Given the constant changes being made to any product in the agile development process, refactoring (rewriting the code) is one of the central practices of agile programming (Larman, 2004)³.

The second motivation is cost. Improving accuracy means spending more time on investigating issues. This time is costly and could be better utilised in the development process itself where the time spent will be contributing towards the improvement of the product.

In essence this pragmatic focus is a form of sceptical empiricism. For those engaging with both sensemaking and agile development, the real issues are practical and observable. Although higher-level discussions and conceptualisation are not discarded, the focus always remains on the stimuli facing the actor in the moment. A byproduct of the focus on the practical and observable is that the focus also shifts to the immediate. Long-term planning is still performed, but decision making is centred around the next step to be taken in the present.

5.2.3 Chaos

The terms *order*, *chaos* and *complexity* are often used colloquially without much consideration for their exact meaning. Before continuing a very brief overview of these must be

³There exists an important distinction between the pragmatic approach of agile development and the idea that code should be refactored to be more ‘elegant’. While some proponents argue against spending time to ‘beautify’ code, other will argue that it is exactly this ‘streamlined’ code that is more efficient and less error-prone.

given⁴.

For an understanding of *order* we turn to March. He lists three ideas comprising the classic conception of order (March & Heath, 1994, p. 49):

1. *Reality* — the idea that there is an objective world that can be perceived and that only one such world exists — history is real.
2. *Causality* — the idea that reality and history are structured by chains of causes and effects.
3. *Intentionality* — the idea that decisions are instruments of purpose and self.

The current environment facing most organisations problematise all three of these classic conceptions. This leads to a context that can be described as *chaotic*. In general vocabulary the term often means to “utter confusion or disorder” (Simpson & Weiner, 1989), but in this case we have to be a bit more specific. Chaos in this context refers to systems that are so sensitive to small changes in conditions that their behaviour seems so unpredictable as to be random (Gell-Mann, 1995, p. 25). This of course breaks with the notion of linear causality and therefore shatters the idea of an “ordered” environment.

Kellert & Snyder (1993, p. 2) provide us with a definition for chaos theory as “the qualitative study of unstable aperiodic behaviour in a deterministic non-linear dynamical system”. Tsoukas (2005, p. 216-217) deconstructs that definition as follows:

- A system is *dynamic* when its state changes over time;
- *Non-linearity* means that a small change in a system variable can have a disproportionate effect on another variable;
- As a result of the difficulty of dealing with non-linear equations, a *qualitative* account of the general pattern of the long term behaviour of the system is sought;
- *Unstable behaviour* means that the system never settles into a form of behaviour that resists small disturbances;
- *Aperiodic behaviour* means that the system does not repeat itself in a regular fashion.

⁴The term “complexity” can have a great variety of meanings depending on the context. This summary aims only to clarify the term in the context of this study.

Table 5.1: *General characteristics of complex systems*

Complex systems consist of a large number of elements which in themselves can be simple.
The elements interact dynamically by exchanging energy or information. These interactions are rich. Even if specific elements only interact with a few others, the effects of these interaction are propagated throughout the system.
The interactions are non-linear.
There are many direct and indirect feedback loops.
Complex systems are open systems — they exchange energy or information with their environment — and operate at conditions far from equilibrium.
Complex systems have memory, not located at a specific place, but distributed throughout the system. Any complex system thus has a history, and the history is of cardinal importance to the behaviour of the system.
The behaviour of the system is determined by the nature of the interactions, not by what is contained within the components. Since the interactions are rich, dynamic, fed back, and above all, non-linear, the behaviour of the system as a whole cannot be predicted from an inspection of its components. The notion of emergence is used to describe this aspect. The presence of emergent properties does not provide an argument against causality, only against deterministic forms of prediction.
Complex systems are adaptive. They can (re)organise their internal structure without the intervention of an external agent.

This chaotic environment is brought about by the complex relations between different actors in the system. We have already referred to Paul Cilliers’s work in the field of complexity, but this discussion would be remiss without a brief summary of the concept *complexity* (Cilliers, 2005, p. 8-9). For that reason the main points are quoted in Table 5.1.

Both agile development and sensemaking occur in an environment that is both complex and chaotic. The salient point here is that both these theories accept this reality about the world, rather than assuming a stable, known context.

Weick et al. (2005, p. 411) explicitly state “sensemaking starts with chaos” acknowledging that it is exactly this absence of order that leads to the conditions that trigger the sensemaking process.

Agile development literature is rife with references to chaos theory as a constituting part of the worldview informing the approach. Schwaber (2004) devotes the fourth chapter of his seminal work to the topic of “Bringing order from Chaos”; Schwaber & Beedle (2002, p. 90) opine “Complexity theory can be seen as a whole new way of understanding the causes and patterns of noise. Noise, it turns out, is an inherent part of everything.” and Beck & Andres (2000) list five sources of information about “emergent processes” dealing solely with chaos and chaos theory.

Most organisations today operate in a space between order and disorder called the “Edge

of Chaos” (Waldrop, 1992) . This zone exist within the sphere of complexity and allows organisations to benefit from the order and regularity that *can* be found there, but also exposes the system to enough unexpected stimuli to prompt creative responses.

In agile development the instinctive reflex of avoiding chaos or shielding the organisation against disorder is replaced with an approach that exposes teams to the rich variety offered by the environment in which they operate.

From a literature review it is clear that notions of complexity and chaos permeate thinking in both agile development and sensemaking. This correlation in fundamental worldview enables a further investigation into the relation between these two approaches.

5.3 Creating Order

The previous section has illustrated that followers of both agile development and sensemaking accept the world to be in a state of chaos. This zone on the “edge of chaos” is a *prerequisite* for the dynamic adaptability displayed in agile development. Sensemaking would also not be possible without the variety of environmental inputs provided by this unstable context.

Although chaos and environmental change are accepted in both cases, agile development and sensemaking are at the same time organising activities. These activities therefore exist in relatively chaotic environs and from this chaos organise processes to help order emerge.

The creation of order is a central motif in sensemaking literature. Weick (2001, p. 95) highlights,

Organising “and” sensemaking turn out to have a closer affinity than is signified by the word “and”. It seems more useful to talk about organising “as” sensemaking, organising “through” sensemaking, or organising “for” sensemaking. To make sense of something is to begin to provide a plausible platform for sharing mental models, coordinating activities, and interacting to produce relationships. To organise around something is to converge on an event whose articulation and preservation feels beneficial and of joint relevance. Sense makes organising possible. And organising makes sense possible.

Agile development is also an organising activity. The very nature of agile development as a methodology (or rather a collection of related methodologies, approaches and methods)

establishes it as a way to introduce order and structure into the development process.

Referring back to our accepted definition of a methodology (p. 11), the structural and organising function of methodologies are apparent. Agile processes acknowledge chaos and change, but provide methodologies as scaffolding which can be used to function within this environment. The chaotic context is therefore not ordered (this would be a futile task), but enough order is created to enable development teams to function. “Scrum asks people to try and wrest a predictable product from unpredictable complexity” (Schwaber & Beedle, 2002, p. 51)⁵.

Agile development is a responsive project, helping development teams cope with the challenges posed by a disordered environment.

5.4 Agile Development as Process

Previous sections have dealt with agile development and sensemaking in some detail. This section will proceed to point out certain important similarities between the two in terms of the actual application of agile development in real-world use cases. Focus is placed specifically on the structural and principle elements enabling an analysis of agile development as sensemaking activity.

5.4.1 Active Construction Process

Agile development is fundamentally a software development methodology (Abrahamsson et al., 2002). Ultimately, the goal, therefore, is to develop a software product. This creative drive is imperative to understanding what agile development is about.

In sensemaking a similar principle is present in two instances. Firstly there is the preoccupation with sensemaking being a creative act of *authoring* (Weick, 1995). This means that sensemaking is not a passive interpretation process that occurs separate from the phenomena under observation. Rather, authoring positions sensemaking as an act in-the-world—an *activity* helping us construct meaning.

Agile development, in the same way, rejects the notion of external analysis and planning, positioning practitioners right in the thick of the problematic context; forcing them to cope

⁵It is important to bear in mind that these organising activities can not be distinguished from their environment and therefore also impact the environment. This enactive effect is discussed in more detail later in this chapter.

with it, struggle through it and *make sense* of it.

Secondly sensemaking places the focus on *action*. This is especially prominent when Weick (1995, p. 13) contrasts interpretation and sensemaking by calling interpretation a “product” whereas sensemaking is an “activity” or “process”. This is typical of the action-driven, interactive nature of sensemaking and is further reinforced by the realistic notion of sensemaking as a way to “cope” with the world. The implication is that sensemaking is not only about understanding the ostensible predicament, but also about *doing* something about it.

Pragmatism is concordantly prevalent in agile thinking. The priority is always tilted towards action and delivery of a real product. Even the second tenet of the agile manifesto (Beck et al., 2001) states that the movement prefers

Working software over comprehensive documentation

The preference for action and development is repeated in the action-driven planning and aims such as “creating a culture driven by performance” (Sutherland, 2007). Agile development thus engages in an active construction process - not only in terms of the product under construction, but also due to the reality construction taking place during the development cycle.

5.4.2 Ongoing Enactment

Sensemaking is an ongoing activity engaged in by all actors. It is not a project or a task, but rather a continuous perspective on constructing and interpreting reality. This is emphasised when looking at sensemaking from the view of enactment theory (Jennings & Greenwood, 2003). This theory constructs a cyclical chain of change, enactment, selection and retention (see Figure 4.1). Repeating their explanation Jennings & Greenwood (2003) state, “sense-making can be treated as reciprocal exchanges between actors and their environments that are made meaningful and preserved”.

There is thus a constant interplay between the environment and the enactment process. The sensemaker has to be highly sensitive to environmental changes and adapt accordingly.

Both agile development and sensemaking rely on some formalised structure to explain the internal mechanics of the processes. Despite most agile methodologies listing a staggered, chronologically sequenced set of steps or phases, the agile development mindset is

continuous. All the underlying assumptions, as well as the worldview, have a major impact on the way actors within agile processes act.

Scrum and XP (the two examples chosen for this study) both employ workflows that contain linear and cyclical elements. The introduction of feedback loops to these workflows embeds these development processes in their environment (in contrast to traditional approaches isolating development from “external influences”).

Agile development is therefore not simply a “project” or “tool” that can be applied in certain circumstances. Once a team selects this approach, it has far-reaching and irrevocable implications. It is not possible to move a development project into an “agile phase”; agile development is an ongoing process deeply influencing the behaviour of individuals engaged in such projects. We can understand why this is the case when reviewing the matter from a sensemaking point of view.

In practice, agile development is also being used for the maintenance of completed products (Svensson & Host, 2005), albeit not without difficulties. This shows that agile approaches are not only ongoing during the initial development process, but may actually be applied throughout the software product lifecycle.

In referencing Heidegger’s concept of “thrownness” (Winograd & Flores, 1986), Weick leads us to conclude that if one accepts that sensemaking occurs in humans, it has to be a continuous, ongoing, *basic* process and that there is no real alternative to this. In fact, the first four points summarised by Winograd & Flores all explain how individuals are compelled to do certain things. (Please refer to the list on page 67.)

Agile development performs in the same way and the point is to allow for an emergent process rather than to contain or limit the process in formalised routines that become abstracted from the context of application and the goals of application. Since planning, adapting, coding and testing are all continuously taking place, the development process is transformed into an ongoing stream of activities with an end-point that is very difficult to determine (if possible at all).

5.4.3 Retrospection

Sensemaking theory proposes that all awareness is awareness of the past; it is not possible to gain access to the immediate present (Weick, 1995). This gives all sensemaking (and by implication all consciousness) a retrospective nature.

Agile development displays the same characteristic and is managed specifically to allow for learning and retrospection as a defined aspect of the process.. Firstly, if you accept the argument forwarded by Weick, agile development (and everything else) would be affected in the same way. This is, however, not the only reason agile development displays elements of retrospection.

Due to the previously discussed heightened sensitivity to feedback, agile development approaches develop several continuous retrospection mechanisms. The idea is that developers should constantly monitor both the environment and the effect their actions are having on said environment.

To this end, teams are not only retrospective, but also introspective. Two of the *ceremonies* in Scrum (Schwaber, 2004), *sprint review sessions* and *sprint retrospective sessions*, deal specifically with looking back at the development process and investigating possible improvements or identifying learnings. These ceremonies happen in addition to the constant critical investigation of work in progress as well as several ad hoc discussions about recent developments.

Retrospective sessions are also important due to their focus on discussion (communication). By talking through the past development sprint, members develop a shared understanding and experience and can process their experiences. This sensemaking process allows them to learn and adapt their own perspectives to foster greater cohesion within the development team.

5.4.4 Adaptive Presumptions

Sensemaking operates through a series of constantly adjusted presumptions, allowing the sensemaker to detect irregular perceptions and flag these for attention. This “series of approximations” (Paget & Cassell, 2004, p. 143) shows an intense adaptability and responsiveness to the environment.

Agile development displays the same pattern during the life cycle of a project. Requirements and specifications are constantly adapted to suit evolving customer needs. In order to manage exceptions and control the development processes, presumptions about the expected state of the system also need to be adjusted.

5.5 Agile Development as Social Activity

Sensemaking presupposes human actors to perform the sensemaking act. This highlights the role of individuals. Since agile development is only applied in situations where humans are cooperating the same applies. This by itself is not particularly surprising or meaningful. However, when viewing agile development as an act of sensemaking, we can come to understand this shift towards the individual.

This shift becomes apparent when comparing agile development to ‘traditional’ development approaches (as was done in chapter 2 and 3). Agile development differs markedly from these approaches in centering activities around human actors (and quite often the individual). Rather than viewing the development process as the primary actor/system, each individual is recognised and valued in terms of his/her contribution to the finished product.

5.5.1 Relation to Individual Sensemaking

The bulk of sensemaking theory (and hermeneutics in general) refers to the coping, understanding, interpretation and *sensemaking* act as it pertains to individual, human actors engaging with the environment and perceptions. This is of course broadened when looking at the sensemaking process of other non-human actors (such as groups or organisations), but it does not negate or preclude sensemaking by the human-actors comprising those structures.

It is important to recognise the fundamentally important role of individual sensemaking, even in organisational contexts. In fact, the social aspects of sensemaking become even more pronounced in social settings such as organisations.

5.5.2 Agile Development as Inherently Social

Beck et al. (2001) list “*Individuals and interactions* over processes and tools” (original emphasis) as the first tenet in the agile manifesto. In breaking with the mechanistic software development methodologies leading up to the declarations in the agile manifesto the primary focus is shifted towards the humans in the development process (developers and clients).

This theme is repeated in principles 4, 5, 6 and 11 (Beck et al., 2001):

4. Business people and developers must work together throughout the project.

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
11. The best architecture, requirements, and designs emerge from self-organising teams.

This level of inclusion and prominence makes it clear that *people* are central to this movement. It also illustrates the striking contrast with modernist organisational theory's view of the worker (programmer) as a functionary on the generic subjective level.

To repeat the quote in chapter three (Highsmith & Cockburn, 2001), “what is new about agile methods is not the practices they use, but their recognition of people as the primary drivers of project success. . . This yields a new combination of values and principles that define an agile world view”.

Scrum

In Scrum the proponents go as far as to list “enhanc[ing] individual development and quality” as one of the main goals of this development approach (Abrahamsson et al., 2002). The benefit of empowering the individual is thus not limited to company but also creates a better value proposition for the individuals involved.

The focus on individuals is not simply a philosophical or ethical aim in Scrum. It does not focus only on the individuals involved, but on the *interaction* between them. This correlates with the emphasis on social elements displayed in sensemaking.

Importantly, it finds expression in the very real, structural conventions (*ceremonies*) in this approach. There are no less than four institutionalised occasions created for social interaction (sprint planning meeting, daily scrum meetings, sprint review session and sprint retroactive session) (Schwaber, 2004). These formal occasions are augmented by the many informal social interactions promoted by the methodology.

eXtreme Programming

When looking at eXtreme Programming (Beck, 2004), several values, principles and practices relate to the social level.

Two of the high-level drivers (values) in the XP process pertain to social matters. The first value listed is *communication*, focusing on the language needed to enable members to work in a team. The last value is *respect* and extolls the importance of respecting each *individual*.

Several of the principles are relevant: *humanity*, *mutual benefit* and *diversity* all deal with ways in which individuals and teams are involved in the development process.

On the practical level (practices) there are many concrete steps taken by an organisation applying XP processes to reinforce the value of the individual and to recognise the social aspects of employees. These include⁶ :

- Sit together
- Whole team
- Energised work
- Pair programming
- Team continuity

From these examples it is clear that agile development also recognises the important role individuals have to play and puts emphasis on the interaction between these individuals (the social context).

5.5.3 Identity Construction

Chapter 4 has highlighted identity construction as one of the key processes during sense-making. Actors engaging in sensemaking do so to restore consistency between their own identity and their perceptions of the world around them. Agile development applies this sensemaking process in two ways.

Firstly, agile development teams are *self-organising* . Being able to determine their own composition and modus operandi is essential to the success of the agile approach (Takeuchi & Nonaka, 1986; Sutherland & Schwaber, 2007). This belief in self-organisation also extends to very practical structural elements, such as the role definitions in Scrum. For instance, there is no leader or external source of authority (such as the Product Owner) who can

⁶Please refer to Table 3.3 on page 52.

interfere in the internal organisation of the group. Through their own autonomy and self-organisation, agile teams are also moulded into tight social units responsible for defining their own identities. The lack of external authority also means that the sensemaking and identity construction process within agile teams are handled as a social, discursive process.

These theoretical deductions are verified by empirical research (Whitworth & Biddle, 2007) finding that real-world agile teams show high levels of cohesion and individuals strongly associate with the team to which they belong. In their findings, the authors (p. 8) also emphasise the issue of identity:

Agile methods were seen in this study to heighten the presence, value, and importance of project team identity as opposed to individual or role-based identity. Constant immersion and engagement with the team as a whole, for example, and the development of rituals surrounding team activity, were seen to support the development and prevalence of a shared identity.

Secondly, the interface between these post-bureaucratic (agile) units and more traditional organisational structures deserves some attention. Agile practices are applied both in small start-up type organisations and in large established enterprises. In both these cases, however, the agile teams generally function removed from traditional organisational structures. This is a result of the vastly different approaches to the development process combined with irreconcilable worldviews. The effect is that these teams often don't align easily with existing defined organisational images and identities. This conflict (dissonance) triggers a sensemaking process leading up to them defining their own identity as a development team. When left without a viable external organisational identity these teams have to resort to giving their own content to the resulting vacuum.

The structure of agile development teams thus creates a trigger for both individual and organisational sensemaking with the aim of constructing an identity for the participants

5.5.4 Organisation through Communication

Sensemaking takes many forms in the organisational setting. One of the easiest to recognise is when groups start organising through a sensemaking process while communicating. This links closely to the social aspects discussed in the previous section.

Any form of groupwork presupposes that the different members of the team will have

some type of communication. In this instance, however, we refer not simply to individuals exchanging data or information. In the sensemaking process events and organisations are “talked into existence” (Weick et al., 2005). This refers to the practice where the acknowledgement of order (or organisation) often occurs through the actions and conversations between constituting actors. By talking “as if” these organisations exist, they in fact construct and reify them. By doing so organisations participate in construction through communication.

The power of communication and conversation consequently makes these interactions between actors of paramount importance to the sensemaking process. This is also seen in the various “vocabularies of sensemaking” identified by Weick et al. (2005).

Agile development recognises the importance of communication. The approach applies several techniques to improve communication and encourage open and frank conversation. Since this type of communication is required for the sensemaking process⁷, the presence of these communication structures in agile development once again makes agile development a candidate for interpretation as a sensemaking activity.

Several practices illustrate this inherent reliance on communication. Nerur et al. (2005, p. 75) go as far as to describe communication as central to agile development,

To summarise, agile development is characterised by social inquiry in which extensive collaboration and communication provide the basis for collective action. Diverse stakeholders including developers and end users go through repeated cycles of thought-action-reflection that foster an environment of learning and adaptation.

Many case studies also report the importance of communication. Whitworth & Biddle (2007) give a particularly insightful view (emphasis added),

... agile participants showed a strong inclination for whole team consideration and involvement. Participants in agile teams would overwhelmingly talk about we and us rather than I or me, and would talk in terms of holistic and systemic visions of the software product or process rather than in terms of individual tasks or roles. Agile participants were seen to be particularly concerned with in-

⁷Or is caused by the sensemaking process due to its necessity.

volvement and inclusiveness across the entire software development team. *Such concerns were supported by agile practices, which allow for ease and speed of communication and collaboration, frequent interaction between team members, and high levels of immersion in project activity.*

Sensemaking literature (for example Obstfeld (2004)) note that it is not only the content of communications, but also the mere *existence* of communication that is instructive in the organising process. To this end the communication function becomes an institutional artefact vital to the organisation (organising) of the organisation itself.

In terms of institutional artefacts, agile development also presents several practices developed to foster communication⁸.

For example, many approaches (such as Extreme Programming) promote *pair programming*. This refers to the practice of two programmers sitting at the same computer and writing code together. Not only does this approach reduce the amount of errors, it also fosters a culture of sharing and interaction between different team members (Williams et al., 2000; Stotts et al., 2003). Implementing pair programming entrenches the notion that development decisions are also social and up for debate and discussion. This emancipates the programmer from the traditional view of software design emanating from an external “architect”. It also debunks the notion that any design is “final” or “authoritative” and opens up feedback loops between the design and implementation functions in agile teams.

Additionally programmers constantly rotate between different pairs. This avoids the formation of cliques and encourages constant mutual learning between different team members.

Agile development methodologies often stress routinised, institutionalised occasions for communication and interaction. In the case of Scrum, these are referred to as *ceremonies*.

In this terminology, a “ceremony” is simply an institutionalised, routinised meeting. Within the standard Scrum process, four *ceremonies* are identified: sprint planning meetings, daily scrum meetings, sprint review sessions and sprint retroactive sessions (Schwaber, 2004). The frequency of these meetings differ depending on the project, but daily scrum meetings are (obviously) always scheduled daily.

These frequent and important meetings reinforce the non-hierarchical nature of agile

⁸Beyond those already outlined in the values and principles discussed earlier in this chapter.

development processes. Since no leader or authority structure is defined, the only way to effect order and synchronise the project is through communication inside and outside the “ceremonies”. Since there will be at least one daily, *formal*, occasion for the whole group to communicate, this becomes the normal way of arranging the functioning of these teams.

Occasions for sensemaking are not the only aspect of communication receiving attention in agile development. The tools enabling and supporting communication as well as the physical spaces used in development are often issues of great interest.

Many software packages have been developed to help manage agile methodologies⁹. These often include visual tools to represent the artefacts (such as burn-down charts, story cards, etc.), and also provide scaffolding for team members to communicate with each other. These are augmented by instant messaging, code comments and message boards. Due to the strong focus on face-to-face interaction these tools are most often used as an augmentation to the existing communication channels. Software tools are also helpful in situations where agile teams are distributed or physically separated.

The physical space in which agile teams work is also designed to foster communication and interaction. Generally programmers all work in an open-plan space with several “common areas” such as lounges, canteens or game rooms. Relaxation and breaks are encouraged both to increase work-life quality and to help provide creative spaces for people to meet and discuss development problems. Marick (2008) expands,

Agile programmers also rely a great deal on constant communication. Teams typically work in bullpens rather than offices or cubicles so that there are no barriers to asking questions or sharing information. Most have daily meetings intended to tell each other what they did yesterday, what they plan to do today, and what help they need.

This also relates to the *concept of ba* introduced by Nonaka & Konno (1999). They contend that *ba* is “a shared space for emerging relationships” (p. 37). By recognising *ba* (place), physical, virtual or mental room is created to aid in knowledge creation. The authors realise that knowledge (also in the development process) is often tacit and needs reflection before it can be exposed in explicit knowledge. Nonaka and Konno argue that

⁹Many of these packages are open-source and available free of charge. A list of these are available at <http://www.agile-tools.net/>. Several commercial products have also been released.

knowledge is embedded in the *ba*. Knowledge can only be explicated from the tacit when actors engage in the *ba* and realise that they are part of their environment.

“To participate in *ba* means to get involved and transcend one’s own limited perspective or boundary. This exploration is necessary in order to profit from the “magical synthesis” of rationality and intuition that produces creativity.”
(p.38)

Without venturing into the field of knowledge management, it is interesting to note that they argue that knowledge embedded in the *ba* only becomes explicit after reflection and experience.

This correlates well with the notion of *articulation* introduced by Weick et al. (2005). In their terms articulation refers to the social process leading up to the transformation of tacit knowledge into something explicit or usable.

Nonaka & Konno (1999) also refer to this *ba* as a “frame in which knowledge is activated as a resource for creation”. Even a casual reading immediately reminds of the concept of “frames” and “minimal sensible structures” in sensemaking literature.

In short the concept of *place* is particularly important to both the knowledge creation and sensemaking processes. Agile development acknowledges this and accommodates it within its communication and organising functions.

Chapter 6

Implications for the Organisation

Right at the beginning of this thesis (see chapter 2) we have seen that agile development differs markedly from traditional software development methodologies. A subsequent sense-making analysis (chapter 4 and 5) has investigated many of these differences from a sense-making perspective. This analysis has also provided a framework with which to explain and understand the real-world, emergent behaviour described as “agile development”.

Agile development as a phenomenon is, however, not simply of academic or theoretical concern. Bear in mind that this is not a hypothetical approach or solution forwarded to try and address current concerns, but rather a real methodological approach being applied in many organisations. This real-world nature prompts a further consideration of the implications of this approach.

This section will briefly deal with a selection of important implications that organisations need to be aware of when considering (or dealing with) agile development. It does not purport to be an implementation guide, but simply aims to illustrate the impact of agile development, especially now that it has been considered as an instance of sensemaking.

The implications listed here are not necessarily unique to agile development and come into play in several contexts where behaviour deviates from the assumptions inherent in current business processes. The small selection identified here aims to serve only as an example in order to illustrate that agile development is not simply a new way of doing “business as usual”.

6.1 Management Challenges

Notions of management and organisation are inextricably linked. Wren (2005) argues that management is a natural evolution of social organising (p. 11):

People found that they could magnify their own abilities by working with others and could thereby better satisfy their own needs. The inputs of various skills and abilities into the group led to the recognition that some were better at some tasks than others. Group tasks were differentiated, that is, there was a division of labour to take advantage of these varying skills. Once labour was divided, some agreement had to be reached about how to structure and interrelate these various work assignments in order to accomplish group objectives. Quite logically, the group also stratified tasks and developed a hierarchy of authority or power.

Without digressing to the field of management practice, it is immediately obvious that agile development stands in stark contrast with some of the central tenets of contemporary management thought. Notions of line functions, division of labour, role description and systemisation all clash severely with the principles of agile development and are still prevalent today¹.

The aim of this discussion is not to argue for a rejection of scientific management principles (Taylor, 1919)², but rather to point out that adopting agile practices necessitates a fundamental re-think of the management approach in any company.

Delegating “planning” functions to higher-level decision makers, studying a specific task or even supervising work become almost impossible in a context where routine and repetition are no longer present.

Chapter 3 has shown that planning and implementing are now intertwined in repeating, recursive cycles. Agile teams are self-organising, determining their own goals and pace and *managing* themselves. In fact, several methodologies (Scrum comes to mind) actively protect the worker from outside supervision or intervention.

¹Consider, for example, the popular concept of organisations as machines (Morgan, 2006)

²This is an extreme example. Of course most companies today utilise a complex mixture of management styles and approaches.

These *radical* changes pose a major challenge to existing managers and management processes in organisations. Boehm (2002) concurs and points out that switching your software development methodology is a complex organisational change that needs to incorporate changes in organisational structure, culture and management practices.

One of the most disruptive implications is that the traditional role of “manager” is replaced with that of a facilitator or coordinator (Highsmith, 2003)³.

Nerur et al. (2005, p. 76) elaborate, “agile methodologies require a shift from command-and-control management to leadership-and-collaboration. The organisational form that facilitates this shift needs to be the right blend of autonomy and cooperation to achieve the advantages of synergy while providing flexibility and responsiveness”.

The notion of a leader is, however, not abandoned; “self-organising teams are not leaderless teams; they are teams that can organise again and again, in various configurations, to meet challenges as they arise” (Cockburn & Highsmith, 2001, p. 132).

The team-nature of agile development also requires companies to modify certain human resource management systems such as “timekeeping, position descriptions... individual rewards, and required skills” (Boehm & Turner, 2005, p. 24). Achievement or merit measurement systems designed for individual workers can in many cases not be used (without major adjustments) in agile teams.

This has profound implications for the expectations managers can have of both the team and the organisational response to the environment. Cockburn & Highsmith (2001, p. 132) summarise

Agile organisations and agile managers understand that demanding certainty in the face of uncertainty is dysfunctional. Agile companies practice leadership-collaboration rather than command-control management. They set goals and constraints, providing boundaries within which innovation can flourish. They are macromanagers rather than micromanagers.

³Compare with the organisational roles defined for Scrum in chapter 3.

6.2 Power and Hierarchy

This new conceptualisation of management as a facilitation role breaks with the more traditional command-and-control approach. The implication is a disruption of entrenched power relationships and authority.

The concept of “power in organisations” is a vast landscape of possible understandings depending on the accepted notion of power (and organisation), the implicit ideologies and the practical case under consideration in any particular discussion⁴. Certain elementary, general implications can, however, be pointed out.

The first is the removal of a structural hierarchy in agile teams. Although certain roles are still defined in approaches such as Scrum, these are not hierarchical in the traditional sense. Individuals appointed in “management” roles tend to be convenors or facilitators, rather than wielding any explicit directing power (at least not by virtue of their position). Although the role of facilitator can certainly be “powerful”, the way it is employed in agile development is as a technical specialist, rather than a manager.

This is in contrast with the view that “hierarchy is a necessary bulwark against disorder, against lower-order members exerting their agency and using power to mess up the rules, task division, and structural designs” (Clegg et al., 2006, p. 135). Agile development rejects the notion of strict “rules, task division, and structural designs” and therefore has no need for these hierarchies or the power embedded in them.

Dismantling hierarchies does not come without consequence. Adapting existing management processes to cope with a new flat-hierarchy is a learning process both for the managers and for the developers.

Empowering individual programmers has the effect of removing certain control and oversight abilities from those managers ultimately responsible for the successful delivery of the product. Without proper adjustments, this leaves the manager unable to effectively (actively) manage his/her project, but still holds them responsible for the ultimate success or failure of the endeavour.

The important distinction is that the application of power (or more specifically *authority*) is very different from that normally expected in organisational contexts. This means that organisational processes and structures have to be able to cope with these discrepan-

⁴For an excellent overview, see Clegg et al. (2006)

cies.

Additionally, this may cause friction when integrating agile development approaches into a larger organisational context where traditional conceptions of management, hierarchy and power are still prevailing.

6.3 People Issues

The previous chapter dealt with the contrast between the traditional conception of “worker” and that adopted by agile development. The effect is that agile development adapts the job and role description to fit with the individual’s skills and interests, rather than simply regarding workers as generic, interchangeable parts, as is the case in many existing approaches.

Boehm & Turner (2005, p. 36) also note this shift

Migrating from traditional to agile management attitudes can be difficult. Large-scale management processes such as earned value and statistical process control evolved from a manufacturing paradigm and tend to cast employees as interchangeable parts. Managers also tend to associate employees with specific roles that might cause difficulty in the multi-tasking characteristics of agile team members.

Since the team is self-managing and no external role definitions are in use, the quality of the people comprising agile development teams is elevated to a deciding factor in the success of any project. Traditional notions of providing workers with formulas or recipes (or even providing extensive training) are rendered useless with the selection of people depending as much on their creativity and problem solving prowess as on their technical skills or knowledge.

This reconceptualisation of the role of the worker puts particular demands on the individuals selected to be part of the team, requiring more competency than is the norm (Boehm & Turner, 2003). Cockburn & Highsmith (2001, p. 131) underline this saying “people trump process” because “agile development teams focus on individual competency as a critical factor in project success.”

Of course the quality of the workers in any organisation is of paramount importance, but in the case of agile development, normal human resource support functions and organ-

isational measures to manage average workers are not applicable. Nerur et al. (2005, p. 76) go so far as to say “at the present time, there is little evidence to suggest that agile principles will work in the absence of competent and above-average people”.⁵

This requirement raises some serious concerns for any organisation electing to implement agile development methodologies. Besides the very obvious staffing issues, integration into the larger organisation is highly challenging when the agile teams are labelled as “elite”. Traditional teams (or other units in the organisation) may be negatively affected by this perception.

The functioning of these teams also require members to cooperate closely and trust each other. Since workers are now recognised as individuals this cooperation cannot be a simple automatic assumption, and the break down of relationships within the team can have catastrophic consequences.

Orlikowski (1992) provides another interesting perspective on the impact these emergent behaviours have on the structure of the organisations by utilising the theory of structuration (Giddens, 1979, p. 64-65): “The theory of structuration recognises that human actions are enabled and constrained by structures, yet that these structures are the result of previous actions”. This theory looks at the social structures in organisations and recognises that actors are knowledgeable and reflexive. Through an interplay of *power*, *meaning* and *norms* the actors themselves become instrumental in determining the structures within which they operate.

This debunks the notion of management as a determining force in organisational design. Companies employing agile development have to take cognisance of this and learn to view even the most fundamental issues in terms of a sensemaking paradigm recognising the individual actors involved.

⁵This is however not a uniformly accepted notion. Boehm (2002, p. 65) remarks, “This is not to say that agile methods require uniformly high-capability people. Many agile projects have succeeded with mixes of experienced and junior people, as have plan-driven projects. The main difference is that agile methods derive much of their agility by relying on the tacit knowledge embodied in the team, rather than writing the knowledge down in plans. . . when the team’s tacit knowledge is sufficient for the application’s life-cycle needs, things work fine. But there is also the risk that the team will make irrecoverable architectural mistakes because of unrecognised short-falls in its tacit knowledge”.

6.4 Knowledge Management Dimensions

Modern organisations consider knowledge a vital factor of production and need to consider ways of managing this “knowledge”⁶. The distributed nature of agile development pose some particular challenges in this regard.

Traditionally organisations rely on generating large amounts of documentation to get a grip on the knowledge they are managing. These records “serve as useful artefacts for communication and traceability of design” (Nerur et al., 2005, p. 76). Since agile teams try to limit documentation as much as possible, this coping strategy cannot be applied. The issue is, however, more complicated and relates to the nature of this knowledge itself.

Tsoukas (1996) reconceptualises the entire firm as a “distributed knowledge system”⁷. He brings this in relation with sensemaking, making this conceptualisation particularly useful for organisations deploying agile development methodologies.

In considering organisations (firms) to be knowledge systems, Tsoukas also explicitly deviates from the neoclassical view of firms and the behaviourist conception of “human agents”. He rejects the notion that individuals are interchangeable or generically substitutable. Instead he acknowledges the view of humans as “agents, active co-producers of their surrounding reality” (p. 13). When rejecting the neoclassical “black-box” view of firms, he also references organisations as “interpretation systems” as an alternative option (Daft & Weick, 1984).

To understand the *distributed* nature of knowledge in organisations, a distinction between *tacit* and *explicit* knowledge is needed⁸. Explicit knowledge is codified and stored in some document or artefact whereas tacit knowledge resides within individuals - often on a subconscious level⁹.

⁶Knowledge Management as a field has developed from work done by many authors, including Nonaka, Snowden, De Jarnett, Clegg, Roos and Chase. The notion of “knowledge management” and its use in practice is, however, not universally accepted and is still a topic of debate in many industry and academic circles. Refer to Wilson (2002) for a more detailed discussion.

⁷Rather than arguing the merits and demerits of this or that definition of KM the field itself, I will refer to a specific argument that is not dependent on an acceptance of KM as a field for its cogency or persuasiveness. Tsoukas fits the bill particularly well.

⁸Tsoukas (1996, p. 14) refines this distinction to a much greater level of detail than is summarised here.

⁹Tacit knowledge in this sense is a necessary enabler or precursor to all knowledge and should therefore not be contrasted with explicit knowledge as two ends of a spectrum. Both forms of knowledge are relevant and exist in an intertwined fashion.

Tsoukas (1996) points us to Nonaka & Takeuchi (1995, p. 61), who argue that “knowledge is created and expanded through social interaction between tacit knowledge and explicit knowledge”. In their model, knowledge is created in five phases (Tsoukas, 1996, p. 14):

The process starts with the sharing of tacit knowledge by a group of individuals; tacit knowledge is subsequently converted into concepts which then have to be justified in terms of the organisation’s overarching mission and purpose; a justified concept is then made tangible, usually through the building of an archetype; finally, new knowledge is disseminated to others within the organisation.

The location of tacit knowledge in several disparate individuals makes the management of this crucial organisational knowledge very difficult. Additionally the distributed nature of the knowledge relates not only to tacit knowledge residing in individuals, but also to the knowledge that exists between them. “*Individual* knowledge is possible precisely because of the *social* practices within which individuals engage - the two are mutually defined” (Tsoukas, 1996, p. 14).

Beyond the distributed nature of *knowledge*, the cognitive processes in organisations are often also distributed (Hutchins, 1996; Hutchins & Lintern, 1995). This means that the entire business process spans more than a single individual (or single mind). When combined with the emergent nature of organisations themselves, this has profound implications.

Building on the metaphor of an individual mind, Weick & Roberts (1993) have conceptualised organisations as “collective minds”. This combines cognitive process, knowledge management and social aspects in organisations, noting that individuals “construct their actions (contribute) while envisaging a social system of joint actions (represent), and inter-relate that constructed action with the system that is envisaged (subordinate).” (p 363).

Tsoukas (1996, p. 17) provides a good summary of the difficulties organisations face when planning knowledge management in an emergent structure:

... all articulated knowledge is based on unarticulated background, a set of subsidiary particulars which are tacitly integrated by individuals. Those particulars reside in the social practices, our forms of life, into which we happen to participate. Before we are cognising subjects we are *Daseins* (beings-in-the-world).

An utterance is possible only by the speaker's dwelling in a tacitly accepted background.

This background to which he refers (unarticulated) is created through a process of socialisation. This is recognised by agile development and is expressed in the pronounced focus it places on the social aspects within the process, team and organisation.

In short, companies operating in the agile development space face enormous challenges in addition to the normal difficulties companies are grappling with when trying to institute institutional knowledge management systems. The emergent configuration of development teams leads to an increased distribution, both in terms of product delivery (cognition) and the knowledge itself. This is compounded by the active reduction in documentation leading to less explicit, codified knowledge artefacts in existence. Traditionally knowledge management systems were only concerned with managing these artefacts and their absence forces organisations to critically rethink these assumptions. Lastly the recognised uncertainty in systems dealing with agile development also raises questions about traditional knowledge management approaches and their suitability given these conditions.

6.5 Compliance and Regulation

All modern organisations have to comply with a variety of legislative, accounting and industry regulation and compliance issues. These compliance requirements (in fact compliance as-such) assumes a certain organisational, systemic regularity and predictability.

In breaking with the formalised business processes present in traditional organisations, agile development makes compliance monitoring and reporting very challenging. Traditional compliance activities rely heavily on measurement and observation (Highsmith, 2002; Boehm & Turner, 2003; Boehm, 2002). Agile development does not have this focus on measuring and is therefore a tough fit with these monitoring processes; “agile methodologies rely on speculation, or planning with the understanding that everything is uncertain, to guide the rapid development of flexible and adaptive systems of high value” (Nerur et al., 2005, p. 77). Boehm & Turner (2005, p. 34) also affirm this: “the level of uncertainty and ambiguity that exists in any evolutionary or iterative process, particularly in long-term estimates, is most likely higher with agile approaches.”

Many industries deploying software projects have to comply with external ratings and

standards set by industry authorities such as CMMI, ISO, DoD, EIA and IEEE. These standards are often not suitable for application in an agile context. Although the quality of the “completed” product may be evaluated, companies deploying agile development cannot comply with the process demands of these certification processes.

Not only are some of the process requirements incompatible with agile methodologies, these specifications normally also require complete audit trails and project documentation to be available. Since the reduction of “reporting documentation” is one of the central goals in agile development, this often leads to irreconcilable incompatibility.

Utilising agile development can therefore present as a legal and audit risk to companies who have to find ways to integrate this flexible approach with very rigid certification demands.

6.6 Coping with these Challenges

This chapter identified a small selection of important challenges posed to organisations deciding to implement agile development methodologies. All of these challenges deserve attention and consideration from senior decision makers.

By demonstrating (in the preceding chapter) agile development to be an instance of sensemaking, organisations struggling with the issues outlined above now have an alternative route of recourse. By considering agile development as sensemaking behaviour, these organisations can gain insights into the mechanics and functioning of the approach, and in so doing develop ways of addressing these concerns.

Since agile development is also only one of a plethora of organisational phenomena within the sensemaking manifold, regarding agile development as part of this selection enables companies to investigate other organisational sensemaking issues and apply some of these insights towards addressing the issues raised by agile development.

Sensemaking also unlocks the underlying worldview central to agile development and in doing so enables key members of the organisation to understand not only the phenomena they observe, but also the principles leading to those actions.

By employing sensemaking, strategic decision makers can evaluate both existing and novel approaches and try and promote integration or collaboration where practicable and appropriate.

Chapter 7

Conclusion

Today companies are more consciously reliant on knowledge and information than ever before. Managing these assets by implementing a variety of information systems has become a multi-billion dollar industry and is affecting all organisations - right from the smallest family-run shop to the largest multi-national corporation. The environment in which businesses operate today is characterised by constant change and inherent dynamism. This change presents a serious challenge to software development methodologies and project management approaches that assume a stable environment conducive to long-term planning.

One of the organic and emergent responses to this challenge of constant change, is the development of alternatives to traditional software development methodologies. These alternatives embrace the complexity of organisational environments rather than trying to explain or reduce it.

This thesis selected *agile development* as one such alternative approach to software development and studied the development thereof as well as its underlying worldview, principles and practices. The aim was to investigate this relatively recent development in order to better understand this phenomenon and its impact on organisations electing to deploy agile processes as a software development methodology.

The factitious nature of agile development elevates this phenomenon from an academic, conceptual topic of concern to a real-world challenge. Organisations today are faced with deploying (or deciding whether or not to deploy) agile development in *real-world* software development contexts. The emergent nature of this shift also compounds the issue due to

the lack of formal literature and academic discourse¹.

Sensemaking theory was selected as a framework with which to try and unlock and understand agile development. Sensemaking builds on the general philosophical theory of hermeneutics². As an approximation, hermeneutics can be understood to describe a “theory of understanding and interpretation of linguistic and non-linguistic expressions” (Ramberg & Gjesdal, 2005).

Sensemaking, however, is more clearly defined than generic hermeneutics. Weick narrows the scope by introducing a cognitive element to the hermeneutic activity. He develops the theory to cover the process of personal sensemaking in some detail and later widens the scope to include sensemaking and interpretation processes occurring within organisations.

By approaching agile development from a sensemaking perspective we were able to gain a better understanding of the mechanics involved in developing software in this fashion. Importantly, however, we also discovered the underlying worldview leading up to and underpinning agile development. This is of paramount importance, since agile development is not a static structure, but rather an emergent representation of a complex set of principles and philosophies³.

Seen in this light, agile development is simply one of a myriad of instances of organisational sensemaking. Some of the insight gained by studying organisational sensemaking (in general) can also be considered as relevant input that can be applied towards managing the agile development process.

7.1 Course of the Research

This study started by investigating the history of software development methodologies. The first step was to define key terms and reflect on the term “methodology”.

Subsequently the chronological history of software development methodologies was investigated in five phases:

- Pre-methodology Era

¹As mentioned earlier, this is of course completely in line with the agile development principle of reducing documentation and favouring functional pragmatism.

²In the tradition of Schleiermacher, Dilthey, Heidegger, Gadamer, Habermas, etc.

³Once again, bear in mind that agile development is principally pragmatic and therefore more concerned with the efficacy of the approach than consciously struggling with the underlying philosophies or ideology.

- Early Methodology Era
- Methodology Era
- Era of Methodology Reassessment
- Age of Agility

These phases trace the development of software engineering as a discipline and the formalisation of programming practice. The evolutionary nature of this development provide this study with the opportunity to investigate certain criticisms on specific methodologies.

Since agile development was established as a reaction to the prevailing methodologies in use, it is important to understand these original methodologies and their development.

The next chapter dealt with agile development itself. Due to the fundamentally pragmatic approach taken by agile development, very little time was spent defining concepts during the founding of this movement. This lack of formal definition created ambiguity and posed difficulties when speaking about “agile development”. To address this, chapter 3 discusses the central principles behind the movement.

Beyond a definition, two practical development methodologies (Scrum and eXtreme Programming) were discussed in some detail. The entire product development lifecycle of a hypothetical product was also studied. Additionally, evidence of the prevalence (and relevance) of agile development in practice was presented.

This chapter formed the working definition of agile development used throughout the rest of the study.

The next chapter introduced sensemaking. Building mainly on the work done by Karl Weick, the term sensemaking was conceptualised, highlighting the historical course leading up to the presentation of this theory. The chapter also presented some of the most important and relevant aspects of the theory, given the context of this study.

An in-depth discussion of the nature of sensemaking was included, covering not only the classic “seven properties” but also a variety of additional characteristics. Special focus was also placed on sensemaking in the organisational setting (in addition to personal sensemaking).

In closing the triggers leading to occasions for sensemaking were discussed.

Chapter 5 proceeded to perform the sensemaking analysis of agile development. Several important characteristics of sensemaking were selected while evaluating the prevalence of

these in agile development. At the same time the reverse was also done in identifying certain key characteristics in agile development and considering these in terms of sensemaking.

The final substantive chapter dealt with the implications of both agile development and this research. Agile development has a big impact on the functioning of organisations and poses many challenges both in terms of management and use, but also in terms of implementation and integration. After identifying these challenges (and exposing the severity thereof) the chapter concluded by explaining the value of a sensemaking understanding of agile development.

7.2 Conclusions

The most important finding was that agile development does indeed display several characteristics that can be better explained from a sensemaking perspective. Chapter 5 has identified several correlations between the principles extolled by sensemaking and the practice experienced when observing agile development in action. To reinforce this, an investigation of the establishment of agile development also shows clear signs of a sensemaking process at work.

To understand agile development it is imperative to realise that this is not simply another alternative in a string of development methodologies. Agile development is *radically* different. Sensemaking provides both the key motivation for the shift and the secret to understanding it.

This study has shown a remarkable correlation between the worldview present in contexts conducive to agile development and that presented by sensemaking. Agile development is thus not simply putting forward a new way of writing computer code, but is much rather a result of the practitioners of this methodology reaching a new understanding of the world in which they operate.

This has profound implications for organisations dabbling in agile development. Agile development is in many ways more a result of the worldview of the developers (or project owner) than an active management decision.

Agile development also differs from traditional software development (or indeed traditional work in general) due to the employees awareness that they are busy with *conscious*

sensemaking⁴. This awareness is tremendously liberating and empowers individual workers to participate in discussions about fundamental issues such as the structure of the organisations of which they form a part.

Through this analysis we come to see that agile development is not simply an ideological project or new programming fad. Agile development is *fundamentally* a pragmatic exercise. It could therefore be argued that the agile approach was developed to address a need resulting from our new conception of the world. When understood in this way, it quickly becomes apparent that you cannot just “plug-in” agile development without exposing the organisation to the whole package of implications of an accompanying worldview.

By recognising agile development as a byproduct of an increasingly complex world and a recognition of the sensemaking activity occurring in both individuals and organisations, companies can understand the real issues raised by agile development. When classing agile development as an act of sensemaking and not a unique phenomenon, they also gain access to many other insights obtained in other processes of organisational sensemaking.

7.3 Further Research

This topic presents several interesting opportunities for further research, especially since very few rigorous academic studies have been done in this field.

The most obvious of these would be a larger scale empirical field-work study applying sensemaking analysis in a practical context. Close observation of the development process would also enable the researcher to calibrate the normative agile practices (used in this study) with the reality experienced in practice.

Several studies focusing on industrial sociology or psychology are also possible. Interesting themes relating to power and authority as well as culture and bureaucracy were identified. Comparing these issues in agile teams with the same metrics in traditional development environments could be very interesting.

Another area deserving more attention is the functioning of decision making in distributed or agile teams. More study is needed to understand the decision making dynamics in these new structures.

⁴Although they may not explicitly know the theory or the terminology associated with it, they are aware that they are engaging in the type of activities and sharing in the worldview described by sensemaking.

It would also be worthwhile to investigate the considerations leading up to the establishment of an agile team. This could also be compared with the success of the projects under investigation. A comparison between agile selection due to requirement versus agile selection as a removed decision (such as a manager reading about this new “style”) could prove very interesting.

7.4 Final Remarks

Agile development is a very real and challenging development facing many companies today. By utilising sensemaking to understand the motivation behind it, rather than dismissing it as simply another management fad, companies can derive great benefits from this approach. Looking at the historical development of programming methodologies, the prevalence of agile development today and the perceived benefits experienced by companies deploying this approach, it seems likely that agile development is here to stay. The radical nature of this approach in contrast with existing methodologies necessitates a novel management approach.

This study has aimed to enhance the understanding of agile development and provide some pointers to help organisations deal with this challenging phenomenon. Realising agile development is both a result and an instance of organisational sensemaking and identity construction will enable companies to accept that project development in this mode will require new ways of thinking about not only software development, but also the organisation itself.

The primary challenge facing companies is to adapt to the reality of radical new approaches (such as agile development methodologies) facing them. By viewing this phenomenon from the perspective of sensemaking, organisations can start to understand that they have a role to play in the management of the conditions under which sensemaking will occur. By supporting, rather than restricting, sensemaking they can increase the agility of the organisation and harness the these already present acts.

Bibliography

- Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). Agile software development methods. *Vtt Publications*.
- Ackoff, R. (1974). *Redesigning the Future*. New York, NY, USA: John Wiley and Sons.
- Ågerfalk, P. J., & Fitzgerald, B. (2006). Flexible and Distributed Software Processes: Old Petunias in New Bowls? *Communications of the ACM*, 49(10), 26–34.
URL http://d.wanfangdata.com.cn/NSTLQK_NSTL_QK12893552.aspx
- AgileAlliance (2010a). Agile Alliance: Agile Certification.
URL <http://www.agilealliance.org/show/1796>
- AgileAlliance (2010b). Corporate Homepage.
URL <http://www.agilealliance.org>
- Ambler, S. (2008). Has Agile Peaked? *Dr. Dobbs Journal*, May08.
- Ashby, W. R. (1956). *An introduction to cybernetics*. London, UK: Chapman & Hall.
- Aspray, W., Keil-slawik, R., & Parnas, D. L. (2007). History of Software Engineering. *Relation*, 10(1.45), 2795.
- Avison, D., & Fitzgerald, G. (2003). *Information systems development: methodologies, techniques and tools*. Berkshire, UK: McGraw Hill, 4th ed.
- Beck, K. (2000). *Extreme Programming Explained: Embrace Change*. Boston, MA, USA: Addison-Wesley.
- Beck, K. (2004). *Extreme Programming Explained: Embrace Change*. Boston, MA, USA: Addison-Wesley, 2nd ed.

- Beck, K., & Andres, C. (2000). *Extreme Programming Explained: Embrace Change*. Boston, MA, USA: Addison-Wesley, 1st ed.
- Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., & Others (2001). The agile manifesto. *The Agile Alliance*.
URL <http://www.agilemanifesto.org>
- Berger, P. L., & Luckmann, T. (1967). *The social construction of reality*. New York, NY, USA: Doubleday.
- Boehm, B. (1986). A Spiral Model of Software Development and Enhancement. *ACM SIGSOFT Software Engineering Notes*, 11(4), 14–24.
- Boehm, B. (2002). Get ready for agile methods, with care. *Computer*, 35(1), 64–69.
URL <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=976920>
- Boehm, B. (2006). A view of 20th and 21st century software engineering. In *Proceedings of the 28th international conference on Software engineering*, (pp. 12–29). ACM.
- Boehm, B., & Turner, R. (2005). Management Challenges to Implementing Agile Processes in Traditional Development Organizations.
URL <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1504661>
- Boehm, B. W., & Turner, R. (2003). *Balancing agility and discipline: A guide for the perplexed*. Boston, MA, USA: Addison-Wesley Professional.
- Brooks, F. (1987). No Silver Bullet: Essence and Accidents of Software Engineering. In *Proceedings of IFIP*, (pp. 1069–1076). IEEE CS Press.
- Burrell, G., & Morgan, G. (1979). *Sociological paradigms and organisational analysis*. London, UK: Heinemann.
- Campbell, D. T. (1987). Blind variation and selective retention in creative thought as in other knowledge processes. *Evolutionary epistemology, rationality, and the sociology of knowledge*, (pp. 91–114).
- Campbell, D. T. (1997). From evolutionary epistemology via selection theory to a sociology of scientific validity. *Evolution and Cognition*, 3(1), 5–38.

- Cao, L., Mohan, K., & Xu, P. (2004). How Extreme does Extreme Programming Have to be? Adapting XP Practices to Large-scale Projects. In *Proceedings of the 37th Hawaii International Conference on System Sciences*, C, (pp. 1–10). Hawaii: IEEE Press.
- Chia, R. (2000). Discourse Analysis Organizational Analysis. *Organization*, 7(3), 513–518.
URL <http://org.sagepub.com/cgi/doi/10.1177/135050840073009>
- Cilliers, P. (2000). What Can We Learn From a Theory of Complexity. *Emergence*, 2(1), 23–33.
- Cilliers, P. (2005). Knowing complex systems. In K. Richardson (Ed.) *Managing Organizational Complexity: Philosophy, Theory, and Applications*. Greenwich, UK: Information Age Publishing.
- Clegg, S., Courpasson, D., & Phillips, N. (2006). *Power and organizations*. London, UK: Sage Publications Ltd.
- Cockburn, A. (2004). *Crystal Clear: A Human-Powered Methodology for Small Teams*. Boston, MA, USA: Addison-Wesley.
- Cockburn, A., & Highsmith, J. (2001). Agile software development, the people factor. *Computer*, 34(11), 131–133.
- Constantine, L. (2001). Methodological Agility: Software Development. *Software Development*, (pp. 67–69).
- Cooley, C. (1902). *Human Nature and the Social Order*. New York, USA: Schribner.
- Daft, R. L., & Weick, K. E. (1984). Toward a Model of Organizations as Interpretation Systems. *The Academy of Management Review*, 9(2), 284–95.
- Deemer, P., & Benefield, G. (2007). Scrum Primer. In J. Sutherland (Ed.) *The Scrum Papers: Nuts, Bolts, and Origins of an Agile Process*, (pp. 20–32). The Scrum Training Institute.
- Dervin, B. (1983). An overview of sense-making research: Concepts, methods, and results to date. In *Proceedings of the International Communications Association Annual Meeting*, (pp. 3–15). Dallas, Texas, USA.

- Dervin, B. (1992). From the mind's eye of the user: The sense-making qualitative-quantitative methodology. *Qualitative research in information management*, (pp. 61–84).
- Dervin, B. (1999). Chaos, order, and sense-making: A proposed theory for information design. *Information design*, (pp. 35–57).
- Dijkstra, E. (1968). Go To Statement Considered Harmful. *ACM Communications*, March, 147–148.
- Duncan, R. B. (1972). Characteristics of organizational environments and perceived environmental uncertainty. *Administrative science quarterly*, 17(3), 313–327.
- Dutton, J. E., & Dukerich, J. M. (1991). Keeping an eye on the mirror: Image and identity in organizational adaptation. *Academy of Management Journal*, 34(3), 517–554.
- Dyba, T. (2000). Improvisation in small software organizations. *IEEE Software*, 17(5), 82–87.
- URL <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=877872>
- Dyba, T., & Dingsoyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9-10), 833–859.
- URL <http://linkinghub.elsevier.com/retrieve/pii/S0950584908000256>
- Erez, M., & Earley, P. C. (1993). *Culture, self-identity, and work*. Oxford, UK: Oxford University Press.
- Favare, J. (2002). Managing requirements for business value. *IEEE Software*, 19(2), 15–17.
- Festinger, L. (1957). *A theory of cognitive dissonance*. Palo Alto, CA, USA: Stanford University Press.
- Fiske, S. T. (1992). Thinking is for doing: Portraits of social cognition from Daguerreotype to laserphoto. *Journal of Personality and Social Psychology*, 63(6), 877–889.
- Fitzgerald, B. (1996). Formalized systems development methodologies: A critical perspective. *Information Systems Journal*, 6(1), 3–23.
- Fitzgerald, B. (1999). An empirical investigation of RAD in practice. In T. Wood-Harper, N. Jayaratne, & B. Woods (Eds.) *Proceedings of the BCS Information Systems Methodologies Conference*, (pp. 77–87). London: Springer-Verlag.

- Follett, M. P. (1924). *Creative experience*. London, UK: Longmans, Green and co.
- Garfinkel, H. (1967). *Studies in ethnomethodology*. Malden, MA, USA: Polity.
- Gasson, S. (1999). A social action model of situated information systems design. *ACM SIGMIS Database*, 30(2), 82–97.
- Gell-Mann, M. (1995). *The Quark and the Jaguar: Adventures in the Simple and the Complex*. New York, NY, USA: Holt Paperbacks.
- Giddens, A. (1979). *Central problems in social theory: Action, structure, and contradiction in social analysis*. Berkeley, CA, USA: University of California Press.
- Gioia, D., & Mehra, A. (1996). A Review of Sensemaking in Organizations. *Academy of Management Review*, 21(4).
- Gioia, D. A., Thomas, J. B., Clark, S. M., & Chittipeddi, K. (1994). Symbolism in and Strategic The Change Academia: Dynamics Influence of Sensemaking. *Organization Science*, 5(3), 363–383.
- Glass, R. L. (2001). Agile versus traditional: Make love, not war! *Cutter IT Journal*, 14(12), 12–18.
- Haungs, J. (2001). Pair programming on the C3 project. *Computer*, 34(2), 118–119.
- Hawrysh, S. P., & Ruprecht, J. (2000). Light Methodologies: It's Like Deja Vu All Over Again. *Cutter IT Journal*, 13(11), 4–12.
- Hedstrom, P., & Swedberg, R. (1998). Social mechanisms: An analytical approach to social theory. *Cambridge: Cambridge Univ. Press. viii*.
- Herbsleb, J., & Moitra, D. (2001). Global software development. *IEEE software*, 18(2), 16–20.
- Highsmith, J. (2000). *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. New York, NY, USA: Dorset House.
- Highsmith, J. (2001). History: The Agile Manifesto.
URL <http://www.agilemanifesto.org/history.html>

- Highsmith, J. (2003). Agile project management: Principles and Tools. *Agile Project Management Advisory Service Executive Report*.
- Highsmith, J., & Cockburn, A. (2001). Development : The Business of Innovation. *IEEE Computer*.
- Highsmith, J. A. (2002). *Agile software development ecosystems*. Boston, MA, USA: Addison-Wesley Professional.
- Hirschheim, R., Klein, H., & Lyytinen, K. (1996). Exploring the intellectual structures of information systems development: A social action theoretical analysis. *Accounting Management & Information Technology*, 6(1/2), 1–64.
- Holford, W., & Ebrahimi, M. (2007). Honda: Approach to Innovation in Aerospace and Automotive/Pick-Up Truck Development: A Dialectical Yet Coherent Firm. In *40th Annual Hawaii International Conference on System Sciences (HICSS-40)*. Big Island, Hawaii, USA.
- Huber, G. P., & Daft, R. L. (1987). The information environments of organizations. In F. M. Jablin, L. L. Putnam, K. H. Roberts, & L. W. Porter (Eds.) *Handbook of organizational communication*. Newbury Park, CA: Sage.
- Hutchins, E. (1996). Learning to navigate. *Understanding practice: Perspectives on activity and context*, (p. 35).
- Hutchins, E., & Lintern, G. (1995). *Cognition in the Wild*. Cambridge, MA, USA: MIT Press.
- IEEE (1990). *IEEE Standard Glossary of Software Engineering Terminology*. IEEE std 610.12-1990.
- Isenberg, D. (1986). The structure and process of understanding: Implications for managerial action. In H. Sims, & D. Gioia (Eds.) *The thinking organization*, (pp. 238–262). San Francisco, USA: Jossey-Bass.
- Jackson, M. C. (2003). *Systems thinking: Creative holism for managers*. New York, NY, USA: Wiley Chichester.
- James, W. (1895). Is Life Worth Living? *International Journal of Ethics*, 6(1), 1–24.

- Jayarathna, N. (2004). *Understanding and Evaluating Methodologies: NIMSAD, A Systemic Framework*. Boston, MA, USA: McGraw-Hill.
- Jennings, P. D., & Greenwood, R. (2003). Constructing the iron cage: Institutional theory and enactment. *Debating Organizations: Point-Counterpoint in Organization Studies*, (pp. 195–207).
- Kellert, S. H., & Snyder, H. (1993). *In the wake of chaos*. Chicago, USA: University of Chicago Press.
- Koskela, L., & Howell, G. (2002). The Underlying Theory of Project Management Is Obsolete. In *Proceedings of PMI Research Conference*. Project Management Institute.
- Kruchten, P. (2004). *The Rational Unified Process: An Introduction*. Boston, MA, USA: Addison-Wesley, 3 ed.
- Kruchten, P., Obbink, H., & Stafford, J. (2006). The Past, Present, and Future for Software Architecture. *IEEE Software*, 23(2), 22–30.
URL <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1605175>
- Larman, C. (2004). *Agile and iterative development: a manager's guide*. New York, NY, USA: Prentice Hall.
- Larman, C., & Basili, V. (2003). Iterative and incremental developments. a brief history. *Computer*, 36(6), 47–56.
- Laudon, K., & Laudon, J. P. (2010). *Management Information Systems*. New Jersey, USA: Pearson, 11th ed.
- Laudon, K. C., & Laudon, J. P. (2004). *Management information systems: managing the digital firm*, vol. 8. New Jersey, USA: Prentice Hall.
- Liker, J. K. (2004). *The Toyota way: 14 management principles from the world's greatest manufacturer*. New York, NY, USA: McGraw-Hill Professional.
- Louis, M. (1980). Surprise and sensemaking: What newcomers experience in entering unfamiliar organizational settings. *Administrative Science Quarterly*, 25, 226–251.
- Lycett, M., Macredie, R. D., Patel, C., & Paul, R. J. (2003). Migrating Agile Methods to Standardized Development Practice. *IEEE Computer*, June, 79–85.

- Maddison, R. (Ed.) (1983). *Information System Methodologies*. Chichester, UK: Wiley Heyden.
- Mailloux, S. (1990). Interpretation. In F. Lentricchia, & T. McLaughlin (Eds.) *Critical terms for literary study*, (pp. 121–134). Chicago, USA: University of Chicago Press.
- March, J. G., & Heath, C. (1994). *A primer on decision making: How decisions happen*. Toronto, Canada: Maxwell Macmillan Canada.
- Marick, B. (2008). A manglish way of working: Agile software development. In A. Pickering, & K. Guzik (Eds.) *The Mangle in Practice: Science, Society, and Becoming*, (pp. 185–202). Durham, NC, USA: Duke University Press.
- Maturana, H. R., & Varela, F. J. (1980). *Autopoiesis and cognition: The realization of the living*. Dordrecht, Holland: R Reidel Publishing.
- McCauley, R. (2001). Agile development methods poised to upset status quo. *ACM SIGCSE Bulletin*, 33(4), 14–15.
- McCracken, D., & Jackson, M. (1982). Life-Cycle Concept Considered Harmful. *ACM Software Engineering Notes*, April, 29–32.
- Miller, G. G. (2001). The Characteristics of Agile Software Processes. In *Proceedings of the 39th Intl Conf. and Exhibition on Technology of Object-Oriented Languages and Systems*, (p. 0385). IEEE Computer Society.
- Milliken, F. J. (1987). Three types of perceived uncertainty about the environment: State, effect, and response uncertainty. *Academy of Management review*, 12(1), 133–143.
- Mills, J. H. (2003). *Making sense of organizational change*. London, UK: Routledge.
- Mills, W. (1959). *The Sociological Imagination*. New York, NY, USA: Oxford University Press.
- Morgan, G. (2006). *Images of organization*. New York, NY, USA: Sage Publications.
- Nandhakumar, J., & Avison, J. (1999). The fiction of methodological development: a field study of information system development. *Information Technology & People*, 12(2), 176–191.

- Nerur, S., Mahapatra, R., & Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. *Communications of the ACM*, 48(5), 72–78.
URL <http://portal.acm.org/citation.cfm?doid=1060710.1060712>
- Nonaka, I., & Konno, N. (1999). The concept of ‘Ba’: building a foundation for knowledge creation. *The Knowledge Management Yearbook 1999-2000*, (p. 37).
- Nonaka, I., & Takeuchi, H. (1995). *The knowledge-creating company: How Japanese companies create the dynamics of innovation*. New York, NY, USA: Oxford University Press.
- Northover, M. (2008). *Towards a Philosophical Understanding of Agile Software Methodologies: A case of Kuhn versus Popper*. Masters thesis, University of Pretoria.
- Norton, D. (2007). Agile Essence: Scrum. *Harvard Business Review*.
- Norton, D., & Murphy, T. E. (2007). Agile Essence : Extreme Programming. *Cycle*.
- Obstfeld, D. (2004). Saying more and less of what we know: The social processes of knowledge creation, innovation, and agency. *Unpublished manuscript, University of California-Irvine, Irvine, CA*.
- Ohno, T. (1982). *Workplace Management*. Cambridge, MA, USA: Productivity Press, Inc.
- Ohno, T. (1988). *Toyota Production System*. Cambridge, MA, USA: Productivity Press, Inc.
- Ohno, T., & Mito, S. (1986). *Just-In-Time*. Cambridge, MA, USA: Productivity Press, Inc.
- Orlikowski, W. J. (1992). The Duality of Technology: Rethinking the Concept of Technology in Organizations. *Organization Science*, 3(3), 398–427.
URL <http://orgsci.journal.informs.org/cgi/doi/10.1287/orsc.3.3.398>
- Paget, M. A., & Cassell, J. (2004). *The unity of mistakes: A phenomenological interpretation of medical work*. Philadelphia, USA: Temple University Press.
- Palmer, S., & Felsing, J. (2002). *A Practical Guide to Feature-driven Development*. Upper Saddle Rivere, NJ, USA: Prentice Hall.

- Parnas, D., & Clements, P. (1985). A rational design process: How and why to fake it. *Formal Methods and Software Development*, (pp. 80–100).
- Poppendieck, M. (2005). A History of Lean: From Manufacturing to Software Development. In *Proceedings of JAOO Conference*. Aarhus, Denmark.
- Poppendieck, M., & Poppendieck, T. (2003). *Lean Software Development — An Agile Toolkit for Software Development Managers*. Boston, MA, USA: Addison-Wesley.
URL <http://books.google.com/books?id=hQk4S7asBi4C&pgis=1>
- Popper, K. R. (1968). Epistemology without a knowing subject. *Studies in Logic and the Foundations of Mathematics*, 52, 333–373.
- Ramberg, B., & Gjesdal, K. (2005). Hermeneutics. In *Stanford Encyclopaedia of Philosophy*. Stanford University Press.
- Randell, B., & Naur, P. (1968). Software Engineering: Report of a Conference Sponsored by the NATO Science Committee.
- Raymond, E. (1999). *The Cathedral and the Bazaar*. Cambridge, MA, USA: O'Reilly.
- Royce, W. (1970). Managing the Development of Large Software Systems: Concepts and Techniques. In *Proceedings of IEEE WESCON*, (pp. 1–9).
- Royce, W., & Royce, W. (1991). Software Architecture: Integrating Process and Technology. *TRW Quest*, 14(1), 2–15.
- Schutz, A. (1967). *The Phenomenology of the Social World*. Evanston, IL, USA: Northwestern University Press.
- Schwaber, K. (1995). Scrum development process. In *OOPSLA Business Object Design and Implementation Workshop*, vol. 27, (pp. 10–19). Citeseer.
- Schwaber, K. (2004). *Agile Software Development with Scrum*. Redmond, WA, USA: Microsoft Press.
- Schwaber, K., & Beedle, M. (2002). *Agile Software Development With Scrum*. Upper Saddle River, NJ, USA: Prentice-Hall.

- Senge, P. M. (1990). *The Fifth Discipline: The Art and Practice of the Learning Organization*. Currency.
- Shannon, C. (1948). A Mathematical Theory of Communication. *Bell System Technical Journal*, 48(27), 379–423, 623–656.
- Shaw, M. (2002). Prospects for an engineering discipline of software. *Software, IEEE*, 7(6), 15–24.
- Shewhart, W. (1939). *Statistical method from the Viewpoint of Quality Control*. Meneola, NY, USA: Dover Publications.
- Simpson, J., & Weiner, E. (Eds.) (1989). *The Oxford English Dictionary*. New York, NY, USA: Oxford University Press, 2nd ed.
- Stapleton, J. (2003). *DSDM: Business Focused Development*. New Jersey, USA: Pearson Education, 2nd ed.
- Starbuck, W. H. (1993). Strategizing in the real world. *International Journal of Technology Management*, 8, 77.
- Starbuck, W. H., & Milliken, F. J. (1988). Executives' perceptual filters: What they notice and how they make sense. *The executive effect: Concepts and methods for studying top managers*, 35, 65.
- Stotts, D., Williams, L., Nagappan, N., Baheti, P., Jen, D., & Jackson, A. (2003). Virtual teaming: experiments and experiences with distributed pair programming. *Extreme Programming and Agile Methods-XP/Agile Universe 2003*, (pp. 129–141).
- Sutherland, J. (2004). Agile development: Lessons learned from the first scrum. *Cutter Agile Project Management Advisory Service: Executive Update*, 5(20), 1–4.
- Sutherland, J. (2007). A Brief Introduction to Scrum. In J. Sutherland (Ed.) *The Scrum Papers: Nuts, Bolts, and Origins of an Agile Process*, (pp. 14–19). The Scrum Training Institute.
- Sutherland, J., & Schwaber, K. (2007). The Scrum Papers : Nuts , Bolts , and Origins of an Agile Process.
- URL http://scrumtraininginstitute.com/home/stream_download/scrumpapers

- Svensson, H., & Host, M. (2005). Introducing an Agile Process in a Software Maintenance and Evolution Organization. In *Proceedings of the Ninth European Conference on Software Maintenance and Reengineering*, (p. 264). IEEE Computer Society.
- Takeuchi, H., & Nonaka, I. (1986). The new new product development game. *Journal of Product Innovation Management*, 3(3), 205–206.
URL <http://linkinghub.elsevier.com/retrieve/pii/0737678286900536>
- Takeuchi, H., & Nonaka, I. (2004). *Hitotsubashi on knowledge management*. New York, NY, USA: Wiley.
- Taylor, F. W. (1919). *The principles of scientific management*. London, UK: Harper & Brothers.
- Taylor, J. R., & Van Every, E. J. (2000). *The emergent organization: Communication as its site and surface*. Mahwah, NJ, USA: Lawrence Erlbaum.
- Truex, T., Baskerville, R., & Travis, J. (2000). Amethodical systems development: the deferred meaning of systems development methods. *Accounting Management and Information Technologies*, 10, 53–79.
- Tsoukas, H. (1996). The Firm as a Distributed Knowledge System: A Constructionist Approach. *Strategic Management Journal*, 17, 11–25.
- Tsoukas, H. (2005). *Complex knowledge: Studies in organizational epistemology*. New York, NY, USA: Oxford University Press.
- Tsoukas, H., & Chia, R. (2002). On Organizational Becoming : Organizational Change. *Organization Science*, 13(5), 567–582.
- Turk, D., France, R., & Rumpe, B. (2002). Limitations of agile software processes. In *Third International Conference on eXtreme Programming and Agile Processes in Software Engineering*, (pp. 43–46). Citeseer.
- VersionOne (2009). State of Agile Survey.
- Waldrop, M. M. (1992). *Complexity: The emerging science at the edge of order and chaos*. New York, NY, USA: Simon & Schuster.

- Wastell, D. (1996). The fetish of technique. *Information Systems Journal*, 6(1).
- Waterman, R. H. (1993). *Adhocracy: The power to change*. New York, NY, USA: WW Norton & Company.
- Weber, K. (2003). *Does globalization lead to convergence? The evolution of organizations' cultural repertoires in the biomedical industry*. Unpublished dissertation, University of Michigan, Ann Arbor, MI, USA.
- Weick, K. E. (1969). *The social psychology of organizing*. Reading, MA, USA: Addison-Wesley.
- Weick, K. E. (1993). The collapse of sensemaking in organizations: The Mann Gulch disaster. *Administrative science quarterly*, 38(4).
- Weick, K. E. (1995). *Sensemaking in organizations*. London, UK: Sage Publications, Inc.
- Weick, K. E. (2001). *Making sense of the organization*. New York, NY, USA: Wiley-Blackwell.
- Weick, K. E., & Roberts, K. H. (1993). Collective mind in organizations: Heedful interrelating on flight decks. *Administrative science quarterly*, (pp. 357–381).
- Weick, K. E., Sutcliffe, K. M., & Obstfeld, D. (2005). Organizing and the Process of Sensemaking. *Organization Science*, 16(4), 409–421.
URL <http://orgsci.journal.informs.org/cgi/doi/10.1287/orsc.1050.0133>
- West, D., Grant, T., Gerus, M., & D'Silva, D. (2010). Agile Development: Mainstream Adoption Has Changed Agility.
- Whitworth, E., & Biddle, R. (2007). The social nature of agile teams. *AGILE 2007*, (pp. 26–36).
- Wiley, N. (1988). The micro-macro problem in social theory. *Sociological Theory*, 6(2), 254–261.
- Williams, L., & Cockburn, A. (2003). Agile software development: it's about feedback and change. *IEEE Computer*, 26(6), 39–43.

- Williams, L., Kessler, R. R., Cunningham, W., & Jeffries, R. (2000). Strengthening the case for pair programming. *IEEE software*, 17(4), 19–25.
- Wilson, T. (2002). The nonsense of Knowledge Management. *Information Research*, 8(1), 8–1.
- Winograd, T., & Flores, F. (1986). *Understanding computers and cognition: A new foundation for design*. Norwood, NJ, USA: Ablex Publishing Corporation.
- Winokur, J. (1990). *Zen to go*. New York, USA: Penguin.
- Wong, C. (1984). A Successful Software Development. *IEEE Trans. Software Engineering*, 3, 714–727.
- Wren, D. A. (2005). *The history of management thought*. Hoboken, USA: Wiley.